Simulating the impact of pairing on Agile software development projects: A multi-agent approach

---

A thesis

submitted in partial fulfilment

of the requirements for the Degree of

Doctor of Philosophy

at

Lincoln University

by

Zhe Wang

Lincoln University

2020

Abstract of a thesis submitted in partial fulfilment of the
requirements for the Degree of Doctor of Philosophy

Abstract

Simulating the impact of pairing on Agile software development projects: A multi-agent approach

by

Zhe Wang

Scrum is an agile process that incrementally, iteratively and continuously delivers software based on time boxes (sprints). It comprises user stories stored in product backlogs and delivered through sprints by a Scrum teams consisting of team members, a Scrum Master and a Product Owner. The performance of a Scrum team is largely dependent on the team members and the technical practices that they adopt. One such practice, pair programming has been studied in a variety of contexts but not extensively in a Scrum context.

Pair programming is a technique where two programmers work side by side at one computer to collaborate on the same design, algorithm, code, or test. The use of pair programming within Scrum has not been widely researched. A multi-agent system is used to simulate the Scrum environment where a team (with varying team members' capability) work on delivering user stories (which consists of multiple tasks with varying complexities) in multiple sprints. Using this simulated environment, various strategies of compulsory pairing and voluntary pairing are investigated. Impact is measured based on the team's work efficiency, completion time, effort time and idle time.

Experiments were carried out to test these strategies in varying environments and results showed that a hybrid pairing strategy performed the best in fixed environments as it avoided negative pairing. An adaptive strategy performed best in the random setting as it was able to use the best strategy based on the current environment. Experiments were also conducted to investigate what happen when there are potential conflicts in the pair within the team. The result suggests that the proposed hybrid strategy was not heavily affected, as even though the rate of conflict increased from 0% to 20%, the work efficiency only decreased by 0.02%. To reflect how these strategies can be implemented in a real world setting, another experiment was conducted using varying sprint durations of 1 week, 2 weeks, 3 weeks and 4 weeks. The result indicates a strong preference towards a sprint duration of 3 or 4 weeks.

Overall, results showed that software development teams using Scrum should use pairing within their set of technical practices for the delivery of software.

Keywords: Scrum, team dynamics, agent-based modelling, multi-agent system, team strategies, solo programming, pair programming

Acknowledgements

During this PhD many unfortunate things happened; however, I overcame them all and I finally achieved this PhD thesis by very hard work. I hereby give thanks to my supervisors, Dr. Patricia Anthony and Dr. Stuart Charters, for their guidance and assistance throughout my PhD study.

In the meantime, I thank my wife, Yanjing Lu, for her support in every aspect, especially for my family.

Publications

1. Zhe Wang. (2020). P-value Based Task Allocation in a Scrum Team: A Multi-Agent Simulation, Institute of Electrical and Electronics Engineers, IEEE Beijing Section,2019 10th IEEE International Conference on Software Engineering and Service Science (ICSESS 2019) held on October 18-20,2019 in Beijing, China.

2. Zhe Wang. (2019). The Compare of Solo Programming Strategies in a Scrum Team: A Multi-agent Simulation Tool for Scrum Team Dynamics. (book Chapter 32) In: Silhavy R., Silhavy P., Prokopova Z. (eds) Intelligent Systems Applications in Software Engineering. CoMeSySo 2019 2019. Advances in Intelligent Systems and Computing, vol 1046. Springer.

3. Zhe Wang. (2018). The Impact of Expertise on Pair Programming Productivity in a Scrum Team: A Multi-Agent Simulation, 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2018, pp. 399-402.

4. Zhe Wang. (2019). Estimating Productivity in a Scrum team: A Multi-Agent Simulation. In Proceedings of the 11th International Conference on Computer Modelling and Simulation (ICCMS 2019). ACM, New York, NY, USA, 239-245. (best paper award)

5. Zhe Wang. (2018). Teamworking Strategies of Scrum Team: A Multi-Agent based Simulation. In Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence (CSAI '18). ACM, New York, NY, USA, 404-408.

6. Zhe Wang. (2020). The Compare of Solo Programming and Pair Programming Strategies in a Scrum team: A Multi-Agent Simulation (Paper ID: 111425) Springer Series: Advances in Intelligent Systems and Computing - ISSN 2194-5357. to appear.

7. Zhe Wang. (2020). Modelling and Simulation of Scrum Team Strategies: A multi-agent approach (Paper ID: 200641) Springer Series: Advances in Intelligent Systems and Computing - ISSN 2194-5357. to appear.

8. Zhe Wang. (2020). Comparisons on Scrum Team Strategies: A multi-agent Simulation. In The 12th ACM International Conference on Computer Modelling and Simulation ICCMS 2020 June 23-25, 2020, Brisbane, Australia, Paper ID: MC1060.

9. Zhe Wang. (2020). Comparisons on Scrum Team Pairing Strategies: A multi-agent Simulation, PAPER ID:D037, is accepted to be published in 2020 the 11th International Conference on

Software Engineering and Service Science (ICSESS 2020) will be held on October 16-18,2020 in Beijing, China.

10. Zhe Wang (2021), Multi-Agent Simulation of Agile Team Dynamics: An Investigation on Team Strategies Perspective is accepted to be published in the springer book Advances in Intelligent Systems and Computing. Springer Series: Advances in Intelligent Systems and Computing ISSN 2194-5357.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Software is created by people using technologies and tools, which are integrated in a process known as the software development process life cycle (SDLC) (Sommerville, 2016). They are two major approaches in software development, the waterfall model and the Agile approach. The waterfall model is a non-iterative sequential process which progresses from requirement analysis through system design, and coding, to testing and maintenance (Sommerville, 2016). Such an approach provides limited interactions between the customer and the software team due to its sequential nature where interactions only happen during requirement analysis.

Agile software development is an approach to creating software through an iterative and incremental process. Beck, Grenning & Martin (2020) describes Agile as: "Individuals and interactions over processes and tools; Working software over comprehensive documentation; Customer collaboration over contract negotiation; Responding to change over following a plan." Scrum is one of the agile processes that realises the Agile manifesto (Beck et al., 2020). It splits users' stories into several parts and aims to achieve each part within a time period, which can last from one day to thirty days, and is known as a sprint (Schwaber & Sutherland, 2017). Within each sprint, the process is similar to the waterfall model. However, there are more opportunities for the Scrum team and the customer (also known as the product owner) to discuss the software requirements during the process because there are multiple sprints in an agile project and each sprint contains requirement analysis and this allows the requirements to be refined during each sprint. Each sprint aims to deliver a higher project success rate compared to the waterfall approach because the size of the software is smaller, its design goal is clearer, and the Scrum team is fully focused on the project (Schwaber & Sutherland, 2017). The Scrum process (Schwaber & Sutherland, 2017) is composed of user stories, tasks, sprints, sprint meetings, deliverable software and conducted by the Scrum team. User stories are designed based on the users' requirements. Each user story contains one or more tasks that need to be completed by the Scrum team during one sprint or across several sprints. A Scrum team comprises several members who have different skills and capabilities, such as designer, developer, database administrator and tester. Scrum team members need to interact and collaborate with each other to achieve the goals of the team, incrementally or iteratively, through sprints. A Scrum team will have daily sprint meeting to discuss what has been done and what needs to be done (Schwaber & Sutherland, 2017).

Although the Scrum approach is an improvement over the waterfall model, it still suffers from several problems, as indicated by the Scrum guide (Schwaber & Sutherland, 2017), which said Scrum is simple

to understand but difficult to master, those including how to organize the scrum team dynamics. One such problem is team dynamics, which largely affect the quality, risk and value of the process. Team dynamics refers to the team's composition, task allocation, interactions between team members and how they work together. Song et al. (2015) defines effective team dynamics according to the following criteria, as also indicated by (Nadler, Hackman, & Lawler, 1979) :

- Team performance (i.e., the product of teamwork meets the expectations of those who use it);
- Member satisfaction (i.e., each team member's experience contributes to his or her personal well-being and development); and
- Team adaptation (i.e., the team experience enhances each member's capability to work and learn together in the future).

A team comprising experienced and highly skilled members will normally perform better than a junior team that is less experienced and skilled. In a Scrum environment, the composition of the team could affect the performance of the team, because a Scrum team needs high levels of cooperation among its team members to achieve the sprint goal. In addition, the skills, experience and capabilities of the team members affect the performance of the team (Schwaber & Sutherland, 2017). Varying methods of tasks allocation may result in different outcomes and may affect the delivery of the software. It is useful to understand how various methods of tasks allocation affect Scrum team performance.

1.1 Scrum Background

This section describes Scrum and its processes in detail. There are three components in Scrum, as shown in Figure 1.1, which are the roles, the events and the artefacts.

Figure 1.1 Scrum model in brief (Justice, 2018)

Roles

There are three roles in a Scrum team: Product Owner, Scrum Master, and Development team (including developers, designers, testers, database administrators, user experience experts). The product owner represents the stakeholders, such as the customers and the marketing manager (Stober & Hansmann, 2010). The product owner must ensure that he or she is representing the interests of all stakeholders. The product owner also provides the requirements, authorises expenditure for the project, and signs off on any deliverable (Schwaber & Sutherland, 2017). The development team is responsible for developing and testing the project deliverables (Schwaber & Sutherland, 2017). Developers usually hold several key skills to solve user story problems, such as programming capability, software system design, software testing and database programming. These skills are matched with a problem-solving background. For example, an Android development would require the developer to have Java programming skills much more than C#. Finally, the Scrum Master (Schwaber & Sutherland, 2017) is responsible for the Scrum process, adjusting the Scrum to best fit the project and the organisation, as well as ensuring that any issue or problem gets resolved so the Scrum team can be as effective as possible.

Scrum Events

Scrum events are referred to as Scrum ceremonies and include, multi-sprint, sprint planning, daily Scrum (stand-up), sprint review and sprint retrospective. At the beginning of the Scrum process, the user stories are defined, prioritised and estimated, then stored in the product backlog (Schwaber & Sutherland, 2017). At the beginning of each sprint, there is sprint planning where the team decides what user stories are worked on during that sprint. Each sprint is, typically, time boxed at two to four weeks. Each day, the team has a daily stand-up meeting (daily Scrum) where they discuss their progress, plan forward and identify problems that may arise. At the end of the sprint, a sprint review is conducted with the product owner to report on progress, to inspect the increments and to discuss anything (if necessary) that may require changes to the product backlog. A sprint retrospective will be conducted at the end of the sprint by the Scrum team with the Scrum Master in order to determine how the previous sprint could have been done better, the defects in the previous sprints and how to improve them (Schwaber & Sutherland, 2017).

Artefacts

The artefacts of Scrum include the product backlog (Alsalemi & Yeoh, 2015), sprint backlog, and increments (Schwaber & Sutherland, 2017). Requirements (user stories) are gathered and prioritised in a list called the product backlog (Stober & Hansmann, 2010). The product backlog is used to hold user stories defined by the product owner and are ordered by their priority. This gives the product owner an advantage in defining the user stories iteratively and incrementally (Mahnič & Hovelja, 2012). The highest priority user story always goes to the top of the product backlog. The Scrum team will then choose the user stories with the highest priorities to work on in each sprint. A sprint can deliver one or more user stories, depending on its size, complexity of the user stories and capability of the team. Several sprints are typically necessary to complete enough user stories for the initial delivery of the software. An incremental delivery model is a key point in encouraging frequent discussions between the team and the product owner. Regular reviews help to guarantee the quality of the product and the increment delivered in each sprint. The process runs iteratively until all the user stories in the product backlog are completed by the Scrum team or the product owner decides to stop the development of the software (Schwaber & Sutherland, 2017).

1.2 Pair Programming

In pair programming, two programmers work side-by-side at one computer to collaborate on the same design, algorithm, code, or test. This kind of programming has been practiced in industry with great success for years. Pair programming can enhance the code quality, learning and knowledge sharing and team working quality. Pair programming can enhance the performance of developers to deliver quality code in less time compared with solo programming. The benefits of pair programming include promoting effective communication within the team, knowledge sharing/transfer, enjoying work,

increase satisfaction and confidence, reduce code defects increase in team productivity and increase in quality of work (Sommerville, 2016).

1.3 Research Motivation

Teamwork is considered as the most critical factor that could affect the successful rate of software projects (da Silva et al., 2013). The Scrum guide indicates that the Scrum team plays an important role in the software development project (Schwaber & Sutherland, 2017). However, the team composition and how the Scrum team should work together (in pair or solo) have not been fully investigated, as there are limited work on pair programming strategies for Scrum team. However there are research done in investigating pair programming performance based on different pairing composition (these related research are described in detail in Section 2.3).

Pair programming is widely used in industry, because it can support two programmers working together, to share knowledge and enhance the software delivery capability (Sommerville, 2016). However, its positive and negative impact on team performance that could affect its software delivery under Scrum context has not been fully investigated. This research investigates the use of pair programming in a Scrum team and how it can affect the performance of the team through multi-agent simulation as there are limited research that focuses on the adoption of pair programming by Scrum teams. A multi-agent approach is adopted in this research to model each developer in the Scrum team and their collaboration. Moreover, there is very limited work using multi-agents system to model Scrum team behaviour, especially on how to use such an approach to address Scrum team strategy design and implementation (Ramanujam & Lee, 2011; Zhen, Wanpeng, & Hongfu, 2018).

The major objective of this research is to investigate the impact of various pairing schemes in a Scrum team. The research will design some metrics to evaluate the performance of these pairing schemes, so as to figure out how different pairing schemes can enhance the Scrum team performance.

1.4 Thesis Overview

This section provides an overview of the thesis. The next chapter, the literature review (Chapter 2), provides a review of the software development processes and modelling that has previously been undertaken. It also provides an in-depth look at pair programming and other work relating to factors that impact software development. It identifies the research gap and outlines the research questions and objectives. Chapter 3 then sets out the various strategies to be explored and the experiments used to test them. Chapter 4 outlines a multi-agent system that has been developed to simulate the Scrum

software development process and to test these strategies in varying environments. Chapter 5 presents the results of the experiments and provides a discussion of them before the thesis concludes with Chapter 6, which also explores avenues for future work.

Chapter 2

Literature Review

This chapter provides an overview of the software development process and describes the Scrum approach to software development in detail. It then reviews the current literature on the software development process modelling of various software development processes, specifically, the Scrum approach. In reviewing the approaches to modelling, aspects of the software development process, team interactions and practices, such as pair programming, are reviewed to establish the current state of the art and highlight the research gaps that exist.

2.1 Software Development Methodologies

In this section, software development processes will be discussed, such as the waterfall model and the Agile approach. The waterfall model was proposed by Royce (1987). Agile methods include the Scrum framework described in the Scrum guide (Schwaber & Sutherland, 2017) as a more recent approach to software development.

2.1.1 Waterfall model

The waterfall model (Palmquist, Lapham, Garcia-Miller, Chick, & Ozkaya, 2013) is one of the process models used in the software development life cycle (Stober & Hansmann, 2010). It is divided into several phases consisting of requirement analysis, system design, system implementation, system testing and further system support (e.g. maintenance). The waterfall model is a linear model in which all the steps happen sequentially, as shown in Figure 2.1.

Figure 2.1 Waterfall model (Stober & Hansmann, 2010)

The waterfall model has been widely used in the software development industry over the last 50 years (Sommerville, 2016). It emphasises that requirement analysis is very important and must be carried out before system design commences. It aims to guarantee that the software delivered is based on the user specifications. However, the software delivered does not always fulfil the user requirements for several reasons (Sommerville, 2016):

- The user requirements were not defined properly.
- The design of the system was incorrect.
- There were defects in addressing complex software system development.
- The system implemented had not been fully tested and verified.
- Further maintenance was expensive and not cost-effective due to the variety of defects.

The waterfall model is recommended when the requirements are well known, clear and fixed (Sommerville, 2016). However, it is commonly acknowledged that user requirements tend to change throughout a project. In this situation, other models can be explored, such as iterative models and Agile (Sommerville, 2016).

2.1.2 Agile software development

Agile was proposed in order to address the defects in the waterfall model, as the waterfall model pays too much attention to processes and tools, comprehensive documentation, contract negotiation and following plans.

Agile is a concept which advocates that people develop software in an agile way, as opposed to the waterfall model (Sommerville, 2016). The agile concept is specified by the following manifesto, which was retrieved from its own website: https://agilemanifesto.org/ (Beck et al., 2020)

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

1. *Individuals and interactions over processes and tools*
2. *Working software over comprehensive documentation*
3. *Customer collaboration over contract negotiation*
4. *Responding to change over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

This manifesto focuses on individuals and interactions, working software, customer collaboration and responding to changes.

Individuals and interactions can provide further understanding within an agile team and the user. Using an agile approach, regular interactions between the development team and the user ensure that all the user requirements are meet, and the quality of the software guaranteed. Agile promotes a self-organised team that enables team members to work together efficiently.

Agile emphasises working software as the goal of the project (Beck et al., 2020). It indicated that not only can the user regularly check the current state of the software but also can provide comments and feedback to ensure that the software fulfils the user requirements. Customer collaboration means that it is more important to understand the user's need than to deliver what was negotiated in the contract. This requires the customer to be involved throughout the software development process and to make sure that the outcome is as desired by the customer (Sommerville, 2016).

Responding to change is an important feature of Agile, it recognises user requirements may change from time to time as its business logic may change or market forces may change the need for a particular feature (Sommerville, 2016). In responding to these changing requirements, the developer team also needs to update its plan in order to capture the updated user requirements.

There are many Agile frameworks that can be used by Agile practitioners. These frameworks include, Kanban (Anderson, Concas, Lunesu, Marchesi, & Zhang, 2012; Cocco, Mannaro, Concas, & Marchesi, 2011; Osama & James, 2015), Extreme Programming and Scrum (Sommerville, 2016). Scrum is a very popular framework and is adopted by many practitioners. Based on the 2016 VersionOne 10th annual state of agile report, 58% agile practitioners use scrum and 10% use scrum/XP hybrid method. Within the framework, Agile practices such as pair programming, planning, stand-ups and sprints (Schwaber & Sutherland, 2017) can be used.

Scrum

Scrum is defined as "A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value." (Schwaber & Sutherland, 2017). Scrum was developed by (Schwaber & Sutherland, 2017) and has been widely adopted and become a common software development method. Scrum is specifically designed to tackle the complexity of software (Schwaber & Sutherland, 2017). Software products can be very complex because of complexity in their development, analysis requirements, technology adoption and functional complexity (Sommerville, 2016). Complexity mainly comes from the user requirements, which need to be addressed by the development team in conjunction with the client representative,

who is the product owner. Scrum has become the predominant approach, and this is explored in more detail in the next section.

The Scrum guide indicates that a Scrum team is a cross-functional team that delivers software based on teamwork (Schwaber & Sutherland, 2017). There are no defined rules about who must work together but the main goal is sprint delivery. The team decides how they should work together in order to achieve the goal. The team can be considered as a self-organised team, which means the team members can organise themselves, including the roles and tasks chosen and sprint planning. In Scrum, self-organising teams make the decisions about sprint planning (Schwaber & Sutherland, 2017). The team needs to decide how many tasks are to be delivered in that sprint and how long the sprint could take and the operations of the sprint planning, including an estimation of the size of the user stories, an estimation of the size of the tasks and the allocation of tasks. In Scrum, the daily stand-up meetings and the sprint retrospective meeting involves only the development team. The product owner should not intervene in their progress unless they are requested to attend the sprint review meeting (Schwaber & Sutherland, 2017).

Kanban

Kanban is a type of agile process that mainly focuses on the speed of development. It considers the progress of each task and the limitations of the people who can work on each task (Lei, Ganjeizadeh, Jayachandran, & Ozcan, 2017).

The following are the basic principles of Kanban for software development:

- Limiting work in process (WIP)
- Pulling value through the development process
- Making the development process visible
- Increasing throughput
- Using a fixed backlog
- Embedding quality

Research studies indicate that Kanban is an easier way to adopt the agile process (Anderson et al., 2012; Cocco et al., 2011; Lei et al., 2017; Lunesu, Münch, Marchesi, & Kuhrmann, 2018; Osama & James, 2015). Features of Kanban such as the visualization of the Kanban board make it easier to be understood by teams who do not have previous experience in Agile. Compared with Scrum, Kanban aims for a low complexity software system delivery. Lei et al. (2017) emphasised that a comparison between Kanban and Scrum should be conducted through a real-world investigation. Research found that Kanban is easier to adopt because it does not focus on team collaborations but focuses more on

task delivery processes, such as design, coding, testing and delivery. The Kanban team does not necessarily have close interactions with each other to share solutions and tackle the complexity of the problem. The Kanban board also supports task quality and delivery efficiency checking; however, it lacks the support of the sprint review and time box constraints for each task. Kanban can support a large number of team members working together and team members can work individually if they want.

Extreme Programming

Extreme programming (XP) is way to enhance collaborative working that aims to improve team performance to produce higher quality software (Beck et al., 2020; Wood, Michaelides, & Thomson, 2013). Extreme programming is a type of agile process that includes several programming strategies, such as pair programming and test-driven programming. It offers the fundamental values of simplicity, communication, feedback and respect. Since extreme programming was proposed, it has provided an ideal opposite to the waterfall model (Sommerville, 2016).

Since Scrum is a popular agile approach, optimizing Scrum team performance through pair programming becomes extremely useful and valuable for the industry and academic. The XP methods such as pair programming has been studied and discussed in Section 2.3, however the research on how pair programming could affect Scrum team performance has not been fully studied.

2.2 Team Dynamics and Scrum Practices

In this section, team dynamics are defined, followed by how Scrum practices may cause problems in the real world based on the literature review, which can provide a broader view of both team dynamics and Scrum practices (de O. Melo, S. Cruzes, Kon, & Conradi, 2013; Sheffield & Lemétayer, 2013; Stober & Hansmann, 2010; Wood et al., 2013). These Scrum mechanisms, such as product backlog, sprint backlog, sprint time boxes, sprint delivery, sprint review and sprint retrospective, all rely on the Scrum team to realise them. Lindsjørn, Sjøberg, Dingsøyr, Bergersen, & Dybå (2016) emphasise that teamwork is the most important factor that impacts project quality. Team members should have good mutual respect, communication, support and cohesion in order to support each other. A self-organised team in Scrum means that the team are responsible for how they work based on the Scrum guide (Schwaber & Sutherland, 2017). The team should work cohesively to achieve the team goals, rather than just think about their individual goals (Schwaber & Sutherland, 2017). A good Scrum team should aim to achieve benefits for the entire team as the highest priority (Schwaber & Sutherland, 2017). Team dynamics refers to team composition, task allocation, interactions between team members and

how they work together. Song et al. (2015) define effective team dynamics according to the following criteria, as previously indicated by Nadler, Hackman & Lawler (1979):

1. Team performance (i.e., the product of teamwork meets the expectations of those who use it).
2. Member satisfaction (i.e., each team member's experience contributes to his or her personal well-being and development); and
3. Team adaptation (i.e., the team experience enhances each member's capability to work and learn together in the future).

The Scrum team is the most critical part of Scrum, especially as the team is self-organising, such that they are responsible for sprint planning and deciding what to do and how to do it. In order to enhance the success rate of a Scrum project, we need to optimise the Scrum team's performance to maximise the team's capability to deliver the software on time.

Moe & Dingsøyr (2008b) indicate that team dynamics are affected by the nature of the team, the personalities of the team, their working relationships and the environment in which the team works. Moe & Dingsøyr (2008b) show that the team's behaviour is very important for the success of Scrum, especially for team leadership, team backup behaviour and mutual trust. The team in agile is self-organising and this provides the flexibility for the team to address its problems (Hoda & Murugesan, 2016). These problems include delayed or changing requirements, lack of senior management at the project level, the challenge of achieving cross-functionality for the individual team members, and task dependency, which challenges the estimation of the task delivery. These existing problems prevent the team from maximizing its performance when using the agile approach. Moe et al. (2010) analysed the teamwork model that defined teamwork components. Their factors included: team orientation, team leadership, monitoring, feedback, backup, coordination, communication, all of which are critical for organizing the agile team.

López-Martínez, Juárez-Ramírez, Huertas, Jiménez, & Guerra-García (2016) indicated problems during adoption of the Scrum approach. Several problems were identified during the adoption of Scrum approach. These problems were distributed among the team, project and aspects of the Scrum process. The team aspects included the people's attitudes, communication, training, efficiency, features, customers, collaboration, workplace, involving the customer, experience, ability to respond, team size, external resources, team diversity and team commitment. The project aspects included rules, the customer, satisfaction, cost, duration, size and complexity, while the Scrum process aspects included results, methodology, simplicity and organization.

Scrum team aspects include various concepts: backup behaviour, problem solving behaviour, trust between team members, team communication, team learning, team skills and capability, working attitudes, team size, ability to respond, responsibility sharing and team diversity (Dingsøyr, Nerur, Balijepally, & Moe, 2012; Dybå & Dingsøyr, 2008; Lindsjørn et al., 2016; Moe et al., 2010; Moe & Dingsøyr, 2008a, 2008b). These aspects affect the teams' performance; however, the most important aspect that can affect team performance in Scrum is task delivery. This involves the task allocation and team composition aspects. Team composition is just a part of team dynamics; however, it is the most important aspect of team dynamics (Dingsøyr et al., 2012; Dybå & Dingsøyr, 2008; Lindsjørn et al., 2016; Moe et al., 2010; Moe & Dingsøyr, 2008a, 2008b).

Team dynamics within the agile environment were initially studied by Ghimire, Gibbs, & Charters (2016). In their paper, they listed factors that can affect the success rate of an agile project, including the people, processes, organization, technical support, project, delivery strategy, agile software engineering techniques, team capability, project management process, customer involvement, customer-centric issues, decision times, corporate culture, control, personal characteristics, social culture, training and learning, project understanding, understanding agile process, team skills, clear communication, customer involvement, organizational leadership and other external factors. These factors can affect team performance and cause people to have difficulty in adjusting to the team strategy and performance.

In their research, Zhou, Kuttal, & Ahmed (2018) list the main factors that affect developers' skill levels and social factors. Skill levels impact how developers can accomplish the task, such that a developer should be capable of using Java to develop a software system that needs Java technology. Social factors impact on how the team collaborates, such as when groups are working together, it is necessary for them to cooperate in order to solve problems. Both these factors are very critical and important and form the term team dynamics, which include task allocations and team composition.

In another study, Stray, Sjøberg, & Dybå (2016) analysed the attitudes of stand-up meetings held during Agile; they undertook a comprehensive investigation at Agile development companies to investigate how the different roles in Agile have different motivations and attitudes at the daily stand-up meeting, especially how those people react to daily stand-up meeting. The research shows that the more software requirement feature oriented roles, such as product owners, team leaders, Scrum Masters, testers, the higher the motivation for the daily stand-up meeting, because the people in these roles are expected to know the progress of the features that are delivered by the developers. However, such roles as software architects and developers are regarded as less passionate and interested in the daily stand-up meeting, because their work focuses more on the system design and implementation aspects, so they are more likely to be resistant to changes in software system features or dynamic

changes in user requirements. However, the more positive motivation for the daily stand-up meeting, the more agile features can be considered by the team (Schwaber & Sutherland, 2017).

Masood, Hoda, & Blincoe (2017) analysed how agile developers consider the factors that will affect them accepting a task. First, the developers all think that the opportunity to learn new technology, tools and domains is the major motivation for them to take a task; secondly, these agile developers prefer to think about the technical complexity of the task; thirdly their technical ability to perform the task will have an effect on task selection. The majority of the agile developers think that the opinion of the manager and the opinions of the team members have little influence on task selection. These researchers state that the developers will only think about themselves during task selection. Based on the above task selection, there is high probability that conflicts can happen between team members, as each wants to maximise their own benefits, without thinking about the others.

2.2.1 Scrum practices

Concerns about the adoption of Scrum have been raised by Ramanujam & Lee (2011), especially the deviations that occur during the adoption of Scrum (Quaglia & Tocantins, 2011). There is potential that people would not fully understand how to practice Scrum, what Scrum is, and what the problems of Scrum are as indicated by these existing works (Perkusich, Almeida, & Perkusich, 2013; Shiohama, Washizaki, Kuboaki, Sakamoto, & Fukazawa, 2012)

Scrum has a simple framework but it is hard to apply (Çetin & Durdu, 2019). Perkusich et al. (2013) indicate that some stakeholders and developers do not understand or are not familiar with Scrum principles, such as: collaboration between the business and development teams, self-organization and incremental product development. Thus, applying Scrum on a software development project never guarantees the project's success compared to applying a traditional approach.

Agile has spread to many organizations. Those research papers in Scrum is widely used in the software industry as a software engineering methodology (Khmelevsky, Li, & Madnick, 2017; Ramanujam & Lee, 2011; Zualkernan, Darmaki, & Shouman, 2008).

Ramanujam & Lee (2011) proposed an agile management framework called collaborative multi-vendor agile scrum software development framework that can be used to manage more people and larger projects. They emphasised that the management layer of the Scrum owner, Scrum Master and Scrum team can be optimised and re-architected to adapt to the Scrum process. This research shows the problems of Scrum in adapting to large scale teams, because Scrum is designed for small teams. The team should not have more than nine people in it and the research is designed for multiple Scrum Masters to support each Scrum team in order to enlarge the Scrum project range.

Some of the Scrum problems identified by Quaglia & Tocantins (2011) are: How to define the task accomplishment sequence? How to manage the product backlog? How the measurement of the product backlog can affect the team performance and task delivery? The authors found that optimization of the Scrum product backlog can maximise the advantages of Agile for dynamic adaption. They proposed to use Scrum to manage and control a simulation project, to maximise the advantages of Agile for dynamic adaptations to produce high-quality simulation projects. The priority of the user stories in the product backlog can be designed to adapt to the Scrum Master's and product owner's demands, because the task accomplishment sequence will affect the value of the software delivered.

However, Shiohama et al. (2012) argued that the length of each iteration in Scrum affects the success of the software project. They also found that the length of each iteration affected the frequency of the customers' and Agile team's interactions, which indirectly affected the accuracy of the user stories. In addition, the length of each iteration affects the efficiency of the Agile team which, in turn, affects the quality and the delivery of the software. In this work, a method to estimate the appropriate iteration length was proposed. Based on this study, it was found that when the user stories change frequently then the sprint length should be shorter and if the complexity of the user stories are higher than expected, the sprint length should be longer. The length of the sprint reflects the number of iterations needed, the more the iterations the shorter the sprint length. The authors found that there should be an optimum number of iterations to achieve minimum cost per iteration as shown in Figure 2.2, as a result of the simulation. Progress in Figure 2.2 refers to the workload undertaken during the multi-sprints for each type of iteration and it shows the progress for per cost are reduced when the number of iterations increases. Less iterations means lower cost.

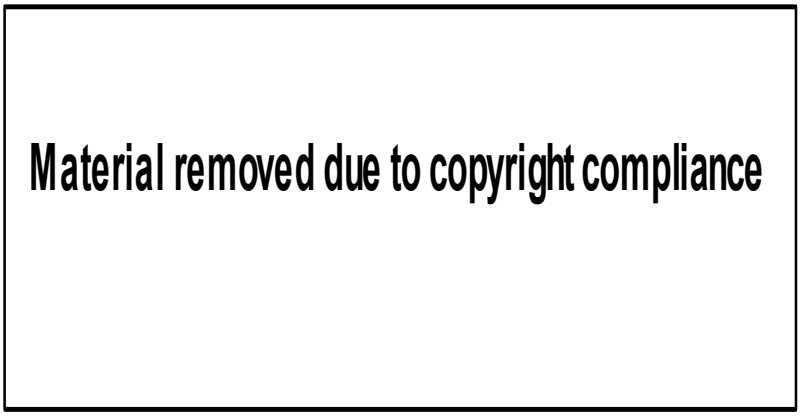


Figure 2.2 Results of the simulation
(Shiohama et al., 2012)

During Scrum practice, some problems could occur such as daily scrum quality and product vision quality as shown in the work of (Perkusich et al., 2013). However, detecting these problems is complex. Perkusich et al. (2013) focused on problem detection in Scrum through graphs, as shown in Figure 2.3. The graph designed is based on personal experience and insights about Scrum. This graph is designed

for data analysis and is useful as it shows the correlations between some of the factors in a Scrum. It presents a probabilistic model to help Scrum Masters apply Scrum in organizations. The goal of the model is to provide information to the project's Scrum Master to help him/her to be aware of the project's problems and have enough information to guide the team and improve the project's chance of success. The graph focusses on monitoring these aspects using: sprint goal checks, stakeholders' feedback, achievements and goals of sprint review meetings, increment acceptance criteria, working validation and its quality, product increment quality, project progress and the overall work quality of the Scrum team.

Figure 2.3 Complete directed acyclic graph designed for the Bayesian network (Perkusich et al., 2013)

Khmelevsky et al. (2017) discovered that distributed team management can affect Scrum performance, when some of the agile projects must be developed through distributed teams. This research focused on the product owner, Scrum Master and Scrum team management. This research shows that the problems in Scrum are because it originally did not support distributed development. It must be adjusted to fit with a distributed team to support distributed software delivery, and this will provide additional complexity when adopting the Scrum framework. It was found that using communication tools was very important, because face-to-face interaction in this context was impossible. However, by using good development tools and online communication, the teams' performance can be improved.

Research paper focused on using computer games were used to train students to understand Scrum (Maxim, Kaur, Apzynski, Edwards, & Evans, 2016). These students became part of a team, where they interacted with the customer in order to make sure the maximum number of user stories can be achieved within time and budget. Different parameters can be set for each game. The purpose of the game is to assist students to understand the correct steps in Scrum. Because most Scrum problems

come from misunderstanding Scrum and the Scrum framework, the Scrum guide does not provide step by step guidance on how to apply it, game-based practice can solve the adoption problems in Scrum. Similar work done by Bassi (2016) indicates game can be used to learn Scrum agile framework.

A study by López-Martínez et al. (2016) revealed that it is not easy for an enterprise to adopt Agile Scrum because the agility of Scrum needs to be comprehensively understood before using it to tackle the problems of the users' changing requirements during the Scrum process. They undertook a comprehensive survey and concluded that the problems for adopting Scrum mainly came from these aspects: people, process, project and organization. The people side included people's attitudes, communication, training, efficiency, features, customers, collaboration, workplace, and involved the customer, experience, ability to respond, team size, external resources, team diversity and team commitment. The project side included rules, customer, satisfaction, cost, duration, size and complexity. The process side also included the results, methodology and simplicity, while the organizational side included culture, management and support.

It was also found that in Scrum, requirement changes affect the user stories, leading to task and product backlog changes which, in turn, affect the sprint backlog (Alsalemi & Yeoh, 2015). The Scrum team needs to be able to react to those changes and update their working plans during the sprints. Requirement changes may result in project delays, additional budget and quality corruption (Alsalemi & Yeoh, 2015). The quality of the software is also part of the user's requirements and this is very important and critical. These changes can also compromise the architectural integrity. The objectives of this research (Alsalemi & Yeoh, 2015) were: to determine how product backlog changes are managed by practitioners in Scrum projects; and to explore the perception of practitioners on the use of requirement traceability for product backlogs and project risk management during requirement volatility in Scrum. This research included a survey among Scrum practitioners and obtained 89 complete responses using an electronic questionnaire. The practitioners identified the reasons for backlog changes and recorded the changed elements. The majority of practitioners recorded modified requirements as a "new requirement," but they also made changes in the original product backlog. The practitioners tended to agree that requirement traceability was helpful in managing the product backlog and minimizing project risks. However, the majority of the respondents pointed out the lack of a specific traceability method for Scrum.

Griffith, Taffahi, Izurieta, & Claudio (2014) were of the opinion that technical debt, such as system errors and defects, needed to be solved before projects can move forward. However, the team is not able to pay too much attention to every debt, otherwise the budget of the project will increase and the project will not be able to be completed within the proposed time and budget. There is a balance in selecting that technical debt. This should be solved as a higher priority compared to implementing

new features, because there are limited resources to be allocated to solve these technical debts and none of the team can solve all these debts during a Scrum process.

Non-functional requirements were regarded as a type of risk and problem. Farid & Mitropoulos (2013) focused on analysing the non-functional requirements and their risk to the Scrum process by developing methods to calculate the types of non-functional requirements and provided an estimation of the potential risk from these non-functional requirements. The two algorithms (risk calculation and schedule computation) they proposed provided the foundation for an improved scheduling technique that can potentially be used by Agile software project managers. This study showed that quantifiable and risk-driven prioritization schemes resulted, potentially, in a shorter, yet more realistic Agile project schedule.

2.2.2 Task allocation research

Task allocation was studied by Masood et al. (2017) who analysed real cases in software companies and found that the majority of developers in the software team considered their skills and matching with the task allocated (Licorish & MacDonell, 2017). The goals of the whole team were often neglected by the team, and this prevented other team members from showing their options and influencing each other in a positive way.

Several studies proposed task allocation algorithm for the Scrum team based on context aware methods (Lin, 2013, 2015; Lin, Yu, & Shen, 2014; Lin, Yu, Shen, & Miao, 2014). This method balances the workload of the task and agent status, such as the requirement for task quality, completion efficiency and an agent's psychology and pressure. Lin, Yu, & Shen (2014) and Licorish & MacDonell (2018) conducted survey-based research about the developer's self-assessment of confidence in completing the task to find whether a correlation between the confidence and completion time of the task is correct or not for task allocation decision-making adjustments. The confidence scale was from 0 to 11 points to indicate not confidence to very confident. Lin, Yu, Shen, et al. (2014) analysed task allocation in the Scrum context by considering the developer's morale. This research had 125 undergraduate software engineering students complete a 12-week course work using the agile software development method. The key findings about the Scrum-based Adaptive Software Development (ASD) process practised by novice teams from this study were: 1) task allocation in agile teams positively correlated with the students' technical productivity; 2) collaboration is negatively correlated with the teams technical productivity, team morale, and team score; and 3) team morale was positively correlated to their technical productivity.

2.3 Pair Programming

Pair programming is a technical development practice proposed in XP that allows two people to work together (Wray, 2010). The two programmers would share the same keyboard and screen to work on the same task at the same time. The benefit of pair programming is effective communication, knowledge sharing/transfer, enjoying work, increase satisfaction and confidence, reduce code defect and increase the product quality and enhance team performance (Sommerville, 2016).

2.3.1 Benefits of pair programming

Based on the work of Haider & Ali (2011), the benefit of pair programming can be categorised into the following aspects:

Effective communication

Pair programming uses two programmers working together, so the chance of frequent conversations is the major method used to improve the understanding of each programmer when solving shared problems. A working environment where two programmers sit at the same computer provides enough time and focus for discussing and exchanging ideas (D'Angelo & Begel, 2017).

Knowledge sharing/transfer

Knowledge sharing and transfer provides insightful thinking about shared problems and helps both programmers increase their understanding about the shared problems (Hagemeister & Rodríguez-Castellanos, 2019; Hanakawa, Matsumoto, & Torii, 2002; Licorish & MacDonell, 2014; Plonka, Sharp, van der Linden, & Dittrich, 2015; Šmite, Moe, Šāblis, & Wohlin, 2017; Williams, McDowell, Nagappan, Fernald, & Werner, 2003; Zieris & Prechelt, 2014). Skills and capabilities can be developed through such a process. A novice pairing can accelerate the learning progress of the novice who gains knowledge from the expert within the limited time of pairing. Plonka et al. (2015) carried out knowledge transfer experiments in pair programming and found that knowledge was transferred during the conversations of the two programmers, as during these conversations the novice can learn from the expert.

Enjoying work

Two programmers working together can provide a lot of fun (Coman, Robillard, Sillitti, & Succi, 2014) in sharing knowledge, learning from each other, and providing a solution for the problem.

Increases satisfaction and confidence

Satisfaction and confidence are shared between the two programmers so they will both become more satisfied and confident when they are paired together.

Code defects and productivity

Pair programming has shown that code defects are reduced (Müller, 2007) and team productivity is increased. Cockburn & Williams (2001) showed that paired programming had 15% fewer defects than solo programming. The paired programming team also produced 20% fewer lines of code than for solo programming. Williams (2002) found that pairs spent 60% more minutes (effort time) on the task but since they worked in tandem, they completed the task 20% faster (elapsed time) than the control groups and produced better algorithms and code.

Quality of the work

The quality of the work is enhanced due to code defects being reduced and productivity enhanced. That good quality work is because of the better solution provided by the pair and the production of high quality code (Nilsson, 2003).

Team performance

As shown from the literature, team performance is enhanced due to the improvements in effective communication, knowledge sharing, transferring, enjoyment of work, increasing satisfaction and confidence, reductions in code defects, enhanced productivity and increased in quality of work.

2.3.2 Pairing performance

Lui & Chan (2006) shows that the performance from combining two novices together is better than one novice alone. However, combining two experts together is not better than a single expert. This result is slightly more advanced than other research, such as the work of Vinod, Padmanabhuni, Tadiparthi, Yanamadala, & Madina (2012), Cockburn & Williams (2001), Williams (2002), Williams & Kessler (2001) and Williams, Wiebe, Yang, Ferzli, & Miller (2002) which explored the performance of pair programming teams.

Lui & Chan (2006) shows an important situation where individuals can perform better than pair programming. The study reports an experiment that repeatedly assigned the same task to individuals and pairs to let them continue working on the same tasks to see how much time was needed. With more time, the pairs all become experts in that task. This is how expert is defined in this paper. The study found that at the beginning of the project, the paired performance was better than solo; however, after the work was repeatedly extended and everyone became expert in the task, it was found that solo performance was better and the solo person learnt even more quickly than pairs through self-learning.

Based on the work Lui & Chan (2006) two experts should not be paired, because their performance is no better than that of a single expert. There can be various reasons for this, such as with two experts

more conflicts may arise than with an expert-novice pair. Novice-novice pairs have less conflict but are provided with more support when doing the task. Vinod et al. (2012) also shows that (Postgraduate-PHD) pair is the best pair compared with PHD-PHD, and Undergraduate- Undergraduate is the weakest pair. This means that the combination of two very high-level experts together (PHD – PHD) is not the best choice. It can work but it can waste team resources and cause potential conflicts between them. However, it also showed that two PHD-PHD as experts performed better than two UG-UG. This suggests that pair programming should only focus on complex tasks, and this is supported by Lui & Chan (2006). If the developer is novice at some tasks, it is better to be paired with another novice, because the paired novice is better than solo novice. If the developers are all experts at the task, it is better not to work in pairs. The high complexity of task can make the pair works better than solo; however, the easy task would not benefit the pair compare with solo. Table 2.1 shows when to pair and why to pair in order to maximise the benefits of pairing. The work by Dybå, Arisholm, Sjøberg, Hannay, & Shull (2007) provided a similar research outcome**.**

Table 2.1 Are two heads better than one? (Dybå et al., 2007)

Guidelines for when to use pair programming			
Programmer expertise	Task complexity	Use pair programming	
Junior	Easy	YES	Provided that increased quality is the main goal
	Complex	YES	Provided that increased quality is the main goal
Intermediate	Easy	NO	
	Complex	YES	Provided that increased quality is the main goal
Senior	Easy	NO	
	Complex	NO	Unless you are sure the task is too complex to be solved satisfactorily by an individual senior programmer.

2.3.3 Pairing impact

The research defines the junior as the novice developer and the senior as the expert developer in the following sections in order to classify the developer team members into three types: novice, intermediate and expert (Arisholm, Gallis, Dyba, & Sjoberg, 2007).

Under Scrum or any other software project, the roles of novices, intermediates and experts are all compared relative to each other or with the task they are working on. Pair programming research indicates that a paired team will normally perform better than a solo developer when they are not skilled enough to complete the task, which means an expert to expert pair is not good. The reason is

because during pairing, they need to cooperate to complete tasks, take advantage of each other's skills, and solve complex tasks based on the less capable individual.

Pairing is most suitable when individuals cannot take on the task. Hannay, Dyb, Arisholm, & Sjobery (2009) indicated that "A more detailed examination of the evidence suggests that pair programming is faster than solo programming when programming task complexity is low and yields code solutions of higher quality when task complexity is high."

As shown by Arisholm et al. (2007) and Hannay et al. (2009), when both paired team members are junior (novice), pairing them together will lead to a higher production rate and enhanced correctness compared with solos. Even though it may take a longer time to complete the task, the quality is enhanced.

If both paired team member are intermediates, then the time used to complete the task by the paired intermediates will be reduced significantly compared to a solo intermediate. The more difficult the task they are working on, the better quality will be gained in comparison with the solo intermediate developer. However, if they are working on an easy task, the quality is reduced and not improved.
However, if both are experts, pairing them together will reduce quality, the correctness rate is reduced, and this expert pairing is considered negative.

Different from same level pairs, a cross level pair is pairing developers that have different capabilities. There has been no direct investigation about cross level pairs based on findings in the current literature but based on the principle of pair programming, two people can work together to learn from each other. If the expert can do the task alone or paired with a novice, then the expert can take the time to train the novice and that is for an intermediate developer as well. These three types of pairing focus more on training and learning purposes and work not only for completing tasks. These cross-level pairings are:

- Novice-expert pair
- Novice-intermediate pair
- Intermediate-expert pair

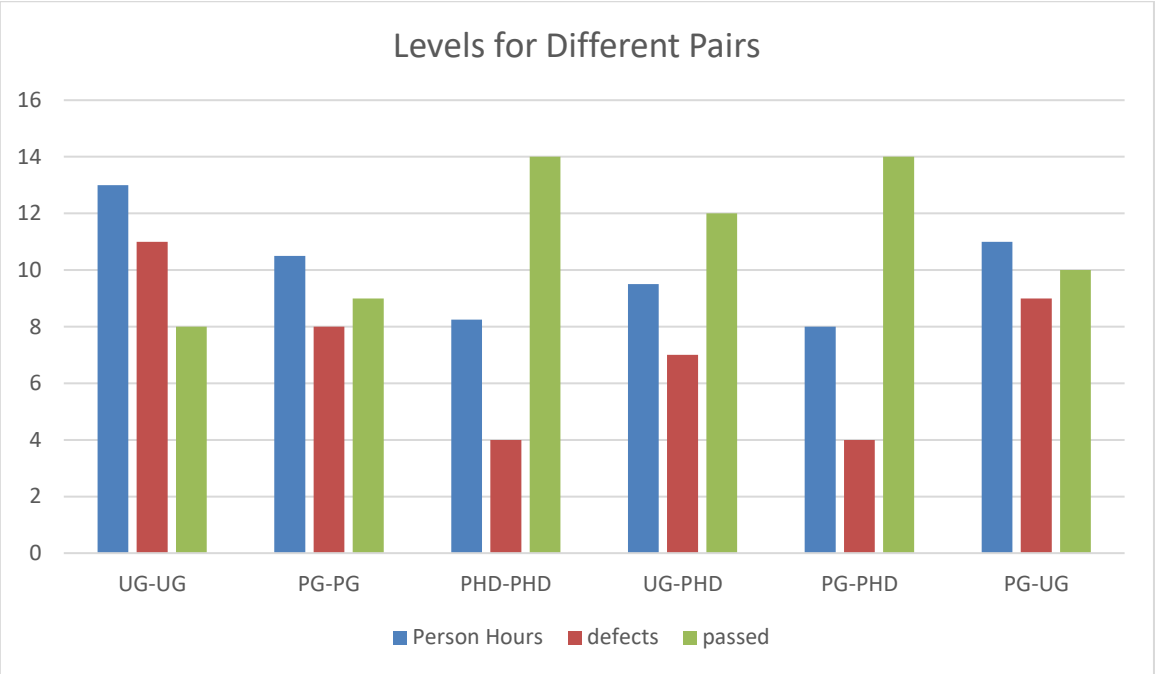


Figure 2.4 Levels of different pairs

Vinod et al. (2012) found that the PHD-PHD and PHD-Postgraduate were the best composition for a paired team, and the performance of PHD-Undergraduate decreased as shown in Figure 2.4. Its performance in passed task is lower than PHD-PHD pair and PHD-PG pair. Based on Figure 2.4, it can be inferred that a novice-expert pair will decrease the experts' performance in number of passed tasks.. However, PHD-Undergraduate is better than Undergraduate - Undergraduate, so a novice-expert pair can improve the novice, which means the novice-expert pair will have a paired team performance between the range [novice, expert]. Postgraduate-Undergraduate did better than Undergraduate-Undergraduate. Postgraduate–Postgraduate's performance was similar to Postgraduate-Undergraduate and Postgraduate-PHD performed better than Postgraduate-Postgraduate. This result infers that the novice-intermediate pair will perform between the range of [novice, intermediate], and an intermediate-expert will perform within the range of [intermediate, expert].

They further concluded that there are no all-time experts or all-time novices as it depends on the relative environment and the context. However, the exact value between [novice, expert] for that pair is dynamics. Not only this value is used to decide the time to complete the task, it also influences how much knowledge can be transferred (from the expert to novice) within that time slot (Vinod et al., 2012).

2.3.4 Related work on pair programming

Cao, Ramesh, & Abdel-Hamid (2010) used a system dynamics approach to model the agile development process to validate the effectiveness of pair programming in an Agile team. It aimed to determine the differences between solo programming and pair programming through modelling and simulation. This study simulated the agile process through system dynamics methods and validated its model through real data collected from real software projects based on solo programming and pair programming methods. The simulation indicated that by using pair programming the quality of the software was enhanced, which reduced the effort of refactoring. The project, which used pair programming, only needed two sessions for refactoring, while the project using solo programming needed three sessions for refactoring, which caused higher costs and effort than pair programming.

Noori & Kazemifard (2015) developed an agent-based model for pair programming in an agile project. They argued that the different personalities and characteristics of the pair will affect the work of the pair programming. They modelled developers through agents and found that some combinations of specific personalities were well suited for pair programming. The results showed that personality played an important role in the formation and utility of a pair. For example, when the expertise of both individuals was high, the best pairing was introvert-extrovert. When both individuals were extrovert, the best pairing was low-high or medium-high expertise. This work focused on the team selection in Scrum.

Pair programming is recommended when task is more complex than the paired team's capability (Nilsson, 2003; Plonka et al., 2015; Williams, 2002; Wray, 2010; Zieris & Prechelt, 2014). Their findings show that pair programming will enhance the team performance more than solo programming. On the other hand, the team's performance is reduced when the pair team's level was higher than the task's complexity. The research on pair programming uses effort time and elapsed time to evaluate the benefit of pair programming. The effort time is the sum of the two developers' working time, while the elapsed time is the time the paired team used to complete the task from start to end.

The study by Williams & Kessler (2001) further provides proof that pair programming can result in high quality outputs and takes less time to achieve. This indicates that paired programmers concentrate on the work and feel confident. The elapsed time compared for one individual and one collaborator is done by the research which indicated that paired can reduce the elapsed time by 40%.

Even though the paired team requires greater effort, the elapsed time to complete the task is reduced with higher quality compared to an individual working on the same task.

A percentage of test cases were conducted based on a comparison of individuals and collaborative teams, and it was found that collaborative teams have higher rates of passing the test compared with the individual cases, due to the high quality of tasks completed.

The impact of pair programming on quality was studied by Nilsson (2003). This study surveyed many related works that others have undertaken and drew conclusions on the relative use of time, which indicated that pairs would use 50% more time than solo in the total time usage. It also indicated that pair programming can enhance the test case pass rate by 20% compared with solo and reduce the lines of code by 20% compared with solo that were written, which shows pair programming is efficient in efficiency and results in better work quality.

2.4 Software Process Simulation and Modelling

Software process simulation and modelling (SPSM) is the simulation of real software development processes using software in order to verify and validate features and factors that could affect the software development process in the most cost effective and convenient ways (Agarwal, 2007; Agarwal & Umphress, 2010; Ali, Petersen, & Wohlin, 2014; Ali et al., 2019; Cherif & Davidsson, 2010). The goal of SPSM is to provide decision making support for managers to make decisions (Joslin & Poole, 2005) in any situation and time in order to build more reliable and complex software for the customer. The variables and parameters are simulated in order to achieve valuable outcomes indicated by Lunesu et al. (2018) These valuable outcomes include the time, cost and quality of the software (Košinár & Štrba, 2013). The quality of software is evaluated by both the functional and non-functional parts of the software system. Different simulations will make use of different variables and parameter settings for different outcomes (Hanakawa et al., 2002).

Managing the software process is uncertain and there are complex interactions between the people involved in that process. These features of modern software development make real case-based process analysis useful. SPSM is a method used to track and analyse the dynamics and interactions inside the software process in order to provide decision making for software process management.

Software process simulation and modelling is used to enhance an understanding of software process management, and includes team management, task management, budget management, project scope, and quality (Agarwal, 2007; Agarwal & Umphress, 2010; Ali et al., 2014; Cherif & Davidsson, 2010). System dynamics and discrete event-based modelling are the two major methods that dominate this research, but agent-based modelling is also increasingly being used. These approaches to modelling and simulation are explored in the following sections.

Kellner, Madachy, and Raffo (1999) provides an overview of the factors as shown in Figure 2.5, that go into the choice of modelling technique and the design of the system.

Figure 2.5 Software process simulation modelling: Why? What? How? Kellner, Madachy, & Raffo (1999)

System dynamics are focused on continuous modelling while discrete events are focused on discrete modelling. System dynamic methods were invented to capture the macro level features of the software process while discrete events were used to capture the micro level features indicated by Kellner et al. (1999). Agent-based simulation is used to capture both micro and macro level features.

Figure 2.6 Taxonomy of simulation techniques(Phillips, 2006)

Phillips (2006) classified the various modelling technologies as shown in Figure 2.6 into three: continuous simulation, discrete simulation and agent-based modelling. This research indicated that agent-based modelling can either use continuous simulation or discrete simulation. System dynamics, which is the most popular simulation method, uses a continuous simulation approach.

For this reason, team dynamics research needs to model human behaviour as its main purpose and only multi-agent based modelling can be used to undertake this research. However, as this method is rarely used in comparison with system dynamics and discrete event-based modelling, it is very hard to find previous experience in modelling Scrum team dynamics through multi-agent based modelling. The use of multi agent system enables researchers to model each individual developer and their behaviour in the team, which is not supported in system dynamics.

2.4.1 System dynamics modelling

System dynamics modelling is an equation-based modelling, which describes the whole system based on the presentation of equations and calculates the dynamics of system behaviour using a macro-view of the system (Wooldridge & Jennings, 1995). Such kinds of modelling ignore the micro view of the system that is concerned with how each factor can affect each other (Wooldridge & Jennings, 1995) .

2.4.2 Discrete event-based modelling

Discrete event-based modelling models a system based on an events-driven process (Gilbert & Troitzsch, 2005). This type of modelling focuses on simulating the events process in the system, rather than other factors. It is used to observe types of events such as busy and idle and the interdependency between them (Gilbert & Troitzsch, 2005).

2.4.3 Agent and agent-based modelling

An agent is a special software component that can produce a self-decision-making process based on its inner rules and logical reasoning (Wooldridge & Jennings, 1995). Agent-based modelling is a way to model and simulate team behaviour, especially groups of people, and their interactions with each other (Macal & North, 2006; Macal & North, 2008). People are keen to use agent-based modelling to carry out social group research, such as team dynamics. Bonabeau (2002) also indicates that agent-based modelling is a powerful simulation modelling technique that has seen a number of applications to real-world business problems, such as flow simulation, organizational simulation, market simulation and diffusion simulation. However, there are limited research with respect to modelling team dynamics in Scrum. The next section explains the status of modelling and simulation on team dynamics for software development process research.

2.5 Modelling the Software Development Process

In this section we focus on modelling approaches that have been used to model Agile software development approaches with a focus on the approaches that have been applied to model the Scrum process. These studies all have purposes, such as control and operational management, planning, process improvement and technology adoption, training and learning (Ali et al., 2014). The scope of these studies involved either a portion of the lifecycle project, the development project, concurrent projects, long term evolution or long-term organization.

Applying software process simulation into software process investigation and research is important (Ali et al., 2014; Cherif & Davidsson, 2010; Hanakawa et al., 2002; Kellner et al., 1999; Zhang, Kitchenham, & Jeffery, 2009; Zhang, Kitchenham, & Pfahl, 2008; Zhang, Kitchenham, & Pfahl, 2010). There are many benefits of software process modelling and simulation and has been applied in operational research and optimization of the current software process by measuring the team size, project costs, task quality, and time and budget.

Simulation project types

These simulation projects are of various types but all fall into several categories, such single phase, multi-phase, projects, multi-projects, products, evolution, long-term organization. The simulation output can be time, effort, quality, size, resource, productivity, cost, benefit, plan, progress, behaviour, flow, change request, human exhaustion level or team's exhaustion.

Simulation research topics

Research topics include generic development, software evolution, software process improvement, requirements engineering, incremental and concurrent development, inspection and testing, open-source development, global development, agile development, software maintenance, software economics, acquisition and outsourcing, software product-line, quality assurance, cost-based development, software engineering education, software design, software services, risk management, productivity analysis and software reliability studies (Ali et al., 2014; Cherif & Davidsson, 2010; Hanakawa et al., 2002; Kellner et al., 1999; Zhang et al., 2009; Zhang et al., 2008; Zhang et al., 2010).

As discussed, there are complexities and difficulties in measuring Scrum (Schwaber & Sutherland, 2017). These measurements can encompass every element of the Scrum framework and they are not limited to team aspects, in order to have a broader view of related work in modelling the development of agile software. The next section will discuss modelling factors regarding Scrum and whether it is team focused or not. This will also include non-agent-based modelling methods as most of the research is not focused on Agile team or agent-based modelling.

2.5.1 Scrum/Agile-based modelling and simulation

The study proposed a reference model about what should be considered in the model from the agile process point of view was proposed by Silva, Rayadurgam, & Heimdahl (2015). In this study, it was suggested that the developer's skills should be considered in the agent model, such as programming skills in Java. The simulation of an agile process can be based on the Scrum model which includes the product backlog, sprint backlog, sprint, the role of Scrum and the activities of Scrum.

Zualkernan et al. (2008) worked on an agent-based simulation project to model Scrum. They simulated the process of Scrum in order to train students to understand the correct steps in the Scrum, such as the role of members and the progress of the Scrum. This training was very useful as it helped these students avoid making fundamental mistakes when applying Scrum in their projects. This is also a type of modelling and simulation that is used for training purpose. This research found that the training system can help people understand what Scrum is and how to practice it, as most of the problems for Scrum occurs during the adoption of Scrum. The research practised the concept of Scrum through the analysis of roles, such as the correct person performing the correct role. For example, who should prepare the product backlog, the Scrum Master or the product owner? For Scrum activity, should the team perform a daily meeting or not? For the sequence of Scrum activity, such as the estimation of the user story points, should it happen before or after the product backlog preparation? For the location of the Scrum activity, should some of the distributed team hold their meetings online? These questions help Scrum practitioners to fully understand the Scrum concepts.

Lin (2013) addressed task allocation in Agile Scrum where he designed an algorithm to allocate tasks to team members and then verified the time delay of the project based on the task allocation methods designed. He found that the task allocation should be both average and focused, which means all team members should be working instead of idling. However, the strong members should not work too much, so as not to cause any further pressure on the others, as this can enhance the team's working efficiency. When the Scrum process is completed, the delay rate of the tasks is calculated and compared to the accept-when-requested task allocations. He found that the algorithm was more stable and performed better for task delay rates when most tasks were completed within the estimated time limitations. The main purpose of this algorithm is to ensure that there should be a balance between the considerations for quality and timeliness when working on the tasks, allocating tasks to highly capable agents, and the on-time completion of the tasks, so there should be low level agents idle. The system was constructed using a multi-agent-based simulation.

Noori & Kazemifard (2015) developed an agent-based model on pair programming for an agile project. They argued that the different personalities and characteristics will affect the work of pair programming. They modelled the developer through agents and found that some good combinations

of specific personalities were well suited for pair programming. The results showed that personality played an important role in the formation and utility of a pair.

Joslin & Poole (2005) proposed several key ideas on how to apply agent-based simulations for software projects. The goal of agent-based software project simulation is for decision support in software project management. The agent can be used to simulate how developers work and how resources are allocated. The agents' strategy can be separated from its algorithmic design. This strategy mainly focused on high level planning while the algorithm focused more on specifics in its realization, such as the agent's behaviour and activity. The search strategy can be optimised using genetic algorithms. There should be initial simulation parameters, which will be updated at each sprint. The duration of the sprint can be defined based on the Scrum Master and the development team. At the end of each sprint, the initially estimated parameters will receive their final value. They also indicated that the task allocation for multi-agents was a critical part of the software development process.

The research by Tamburri, Razo-Zapata, Fernández, & Tedeschi (2012) explores various time zone-based Scrum and how time can affect task completion as, obviously, the time zone will affect the tasks assigned to the team who can be in different locations (global software engineering). Delays or errors in the tasks caused by communication are affected by the time zone, which needs to be considered and optimised through Scrum and adapted to be distributed for worldwide team performance. The stakeholder is also located in a different location and they have no information on the whereabouts of the team members. These tasks are dependent on each other, which makes the delivery process even more complex. The time zone gap is one of the major difficulties that prevents team members from communicating effectively and in a timely fashion. An error occurring in one task may be informed about, or detected by, a team in another place, so two teams can be affected by this error that was shared between tasks. The research further adopts the scrum and agile service network simulation to understand the performance of the model and found that the agile service network is more feasible in supporting global software engineering.

The study by Griffith et al. (2014) provides a discrete-event based simulation of the Scrum process with a particular focus on defects and technique debt creation. They updated the Scrum process by considering technical debt creation. The purpose of this simulation is to investigate how technical debt could affect the Scrum process, including the impact on the software products delivered, as the team effort is a limited resource. However, the technical debt can be almost unlimited depending on the type of debt. The estimation of the time needed to complete a task is based on the developer's capability, where juniors need 2.0 time, middle level developers need a 1.0 time and seniors need 0.5 time to complete a given task. This shows that the more technical debts the lower the team performance. If the error size is the same as the system size, then the team productivity is zero. The

smaller the error size the higher the team productivity. This study focusses on the effect of technical debt in Scrum. It further simulates the debt process into the Scrum process, so technical debt needs to be solved before the project can move on. However, the Agile team cannot pay too much attention to every debt, otherwise the budget of the project will be increased.

The research by Lunesu et al. (2018) used an event-based simulation to model and simulate software project delivery through Scrum. They compared the size of the project and the number of developers to predict the effort and time to complete a project through event-based simulation as well as the differences between the Scrum and Kanban models in delivering software through the cloud platform. They found that project teams faced problems with the communication and organization of distributed projects that affect teams' productivity and increases the time required to achieve the project's goals.

Orłowski, Bach-Dąbrowska, Kapłański, & Wysocki (2014) proposed a framework to simulate the Scrum process. The system simulates the Scrum methodology, including its management processes and project roles. For the implementation of Scrum processes, a Scrum ontology is proposed for the competencies of the roles of project team members and a fuzzy-logic representation is provided. As a result, they presented the hybrid fuzzy-ontological system. The framework of the design processes proposed in the article was verified based on project management processes in a large IT company.

Košinár & Štrba (2013) used an event-based simulation to provide an estimation for the time and effort needed to complete healthcare information projects through Scrum modelling and simulation. They simulated the number of tasks and the size of the development team in order to estimate the completion time of the project using a burndown chart. Even though the simulation was very close to the real world, not all simulations are precise enough due to external factors such as unpredicted obstacles or changes to the teams.

2.5.2 Other non-Agile based modelling and simulation

There are other related works that focus on teamwork or software development process in a non-agile setting.

Spasic & Onggo (2012) used diagrams to represent the behaviour of the developer and the component agents. The developer agent has three roles: unassigned, working, and waiting for prerequisites. The study modelled the task as the component agent, which also has three different states: there is work, waiting and complete. Their main research focus was on using agent-based modelling to simulate an existing development process to see if that simulation fits with the real world. The aim of the work was to use agent-based methods to simulate the software development process. They advocated that a simpler model had better simulation results and was feasible in some situations. They used two types of agent to simulate the software development process: a developer agent and a component agent.

The component agent contains the basic state transfer from $S \rightarrow S'$. The developer agent only contains the state of transfer but does not have further complex ability for reasoning, collaboration or communication as humans do. The component agent contains task specifications, such as task size and task completion status, while the developer agent contains working status and activities associated to these tasks. The simulation model used the waterfall model in software development but ignored interactions between developers and task allocations during the development process. This model can provide an overview on the progress of the project but cannot provide detailed information about how the project is affected by the team and the tasks it works on.

Multi-agent simulation with system dynamics methods were compared by using several mathematical computational models to describe the developers' performance (Cherif & Davidsson, 2010). They also used those models to calculate the size and quality of the software for the performance and time spent by each developer to develop the final product. They compared the performance of the system dynamics and multi-agents. In a single agent comparison, the average value of that agent was the same as in the system dynamics model (both the agent-based modelling and system dynamics modelling shared the same personal performance calculation model). The single agent behaved in a very similar fashion as in the dynamics system of modelling, while the multi-agents behaved differently from the system dynamics model. The single agent could not describe the behaviour of each individual task, while the multi-agent was able to describe the behaviour of each individual task, and this can be used to capture the variance between individuals.

Andrejczuk (2018) researched on team decomposition where the aim is to split a large group of people into many small groups and these small groups can then perform optimised behaviour. When splitting large groups, there should be an even distribution in the number of experts, intermediates and novices in the small groups so that the performance of these small groups is not compromised. The members of these small groups are selected through skill matching to ensure that they can perform all the tasks assigned to the groups. However, how those small group can work on those tasks are ignored and not researched.

The role of personality as well as the projects' parameters using the MBTI (Myers-Briggs Type Indicator) to identify and represent the personality of a person was investigated by Noori & Kazemifard (2015). Software agents were then used for simulating pairs. There were two kinds of agents in this simulation: team member agents (TMA) and simulation agents (SA). TMA is a fuzzy agent for simulating a team member. A team can be simulated by multi-TMAs. Each TMA is characterised by five internal variables: introversion/extroversion, sensing/intuitive, feeling/thinking, perceiving/judging and problem difficulty. A multi-agent was used to simulate intrateam communications. The model was evaluated using five modes. The results show that personality plays an important role in the formation and utility

of a pair. For example, when the expertise of both individuals is high, the best pairing is introvert-extrovert. When both individuals are extrovert, the best pairing is low-high or medium-high expertise.

Agarwal (2007) simulated an existing software development process and compared the differences between the actual process and the simulated process. The more similar the two, the better the agent-based simulation model is. The estimated value, Personal Software Process (PSP) data is the initial value that triggers the simulation process. There are two types of values, the simulated value and the actual value. The actual value is a pre-existing value based on the same PSP data collection, while the simulated value is obtained from the simulation model. The comparison is based on the differences in these two values for each project. The four developer group PSP data for the four projects are collected. Each project will result in a simulation value and this value was compared with the pre-existing actual value. The actual and simulated data were plotted on a line chart and a very similar trend was observed between the actual and simulated data as for the results of the simulation. The more similar the two values were the more accurate the simulation model.

Team behaviour that can affect the software development process was studied by Phillips (2006) which uses four types of teams: synchronised teams, concurrent teams, agile teams and autonomous teams. Synchronised teams approached the problems in a linear, top-down fashion. Concurrent teams approached the problems in a concurrent, top-down manner. Agile teams approached the problems in a linear, bottom-up fashion. Autonomous teams followed a concurrent, bottom-up approach. The aim of this study was to evaluate how different types of team organization can affect the stability of productivity, stability of staff utilization, stability of timeliness, stability of quality and the relations between stability ,quality, turbulence, productivity, turbulence, staff utilization, timeliness, turbulence and quality, turbulence here means problems occurs by team. The agent was modelled to represent an entire team, which ignored intra-collaborations within a team and only focused on the interactions between teams. The study found that concurrent teams were the most affected by turbulence, Agile teams were most suitable for small teams and projects, while autonomous and concurrent teams were suitable for large teams and projects.

A general simulation framework was proposed by Uzzafer (2013) and compared various strategies for software project management. These strategies were used for software project cost estimations and planning. Three aspects were analysed: risk management, cost estimation and project management. Uzzafer (2013) developed several methods to calculate these values based on the strategies chosen. The results showed that the project was able to compare the strategic impacts on these three aspects, in which different strategies result in different risk management, cost estimations and project management for different projects. The proposed simulation model is generic, so it has generic components with plug and play interfaces, which allows any sets of risk assessment and cost

estimation models to be used and any project management planning tools to be adopted in the construction of the simulation model. The model is designed to be user-friendly such that academics and practitioners can easily adopt the proposed simulation model for further research and development.

2.6 Summary of the Gap

Based on the literature review, there is very limited research on pair programming and its impact on the performance of Scrum teams, as this research has not been fully investigated. Study on team dynamics in Scrum, especially for task allocation and team composition, are not fully understood under the context of the Scrum framework, which limits the practices of teams working on adoption strategies by the Scrum team.

Meanwhile, Scrum team modelling and simulation through multi-agent methods have not been fully investigated either, so this also limits the design of teams working on strategies under Scrum rules. Based on the literatures, we have identified the following gaps:

1. There is little modelling and simulation that focuses on team dynamics research particularly in a scrum environment.

2. Pair programming is a type of team dynamics that has many benefits (Sommerville, 2016). However, such positive benefits have not been reflected in Scrum teams through a defined strategy. This strategy should enhance Scrum team performance through pair programming to show its positive impact on team performance that can accelerate the team to achieve their Scrum goals at each sprint. The pairing of different developers should consider the impact on each other and also the efficiency of task delivery.

3. Current modelling and simulations have undertaken very limited work on real multi-agent-based modelling and simulation for Scrum teams. This causes many restrictions on this type of research. Good team strategies can only be designed through real multi-agent modelling so it can capture each agent's own preferences for task selection and pairing.

4. Even though we have known that pair programming or solo programming can be practised in the Scrum team, there is no such strategy that can always optimise the performance of the Scrum team. Various strategies should be developed to guide how to practise solo or pair programming in a Scrum team, in order to maximise team performance, especially for sprint delivery.

2.7 Research Questions

To address those gaps, we have formulated 4 research questions:

1. How does the adoption of pair programming impact the effort required to implement a software project?

2. What impact does the chosen pairing scheme have on the effort required?

3. How does team composition impact project completion under different pairing schemes?

4. In which situations is pairing advantageous compared with not pairing?

Based on those research questions, we would like to investigate the pairing and its impact on team effort, such as efficiency , completion time and idle time. These measurements will be detailed in the next chapters. It is possible that various pairing schemes may performed differently depending on the environment it is in, and this will be investigated in this research. These pairing schemes and solo programmming could be compared to see which is more advantageous under various circumstances.

In this chapter, we discussed the related work on Scrum modelling, pair programming and various approaches to simulation and modelling processes. We also described the research gaps and presented a list of research questions that we will attempt to answer in this thesis. In the next chapter, we will describe the various pairing strategies that we will implement in order to answer these research questions.

Chapter 3

Method

In this chapter, we describe a solo strategy that can be used in a situation where a single task must be worked on by a single agent. We also describe two pairing strategies where a pair of agents must work as pairs to complete a task (must pair) and where a task may be worked on by a pair of agents or a single agent (voluntary).

3.1 Strategy Design

To evaluate the impact of pairing in a Scrum software development project, we developed one solo strategy and two pairing strategies (must pair and voluntary pairing) that can be used by the Scrum team to complete all the user stories in the sprint backlog. The must pair dictates that the agents must work in pair to complete a task whereas with voluntary pairing a task may be worked by a single agent or a pair of agents. The purpose of introducing these two types of pairing is to investigate the impact of compulsory pairing as opposed to voluntary pairing. Voluntary pairing is designed to maximise the benefits of pairing, so we only pair agents when it is beneficial to pair as this allows them to work solo when no acceptable pair is found. We further sub-divided the must pairs and voluntary pairs into three sub-categories based on the capability of the pair.

These strategies are designed to be evaluated in a single sprint where a sprint contains multiple user stories which are further subdivided into tasks. Each task has a size which ranges from 1 to 10, where 1 is the easiest task and 10 is the most difficult task. Tasks between 1 and 4 are categorised as easy tasks, 5-7 are intermediate tasks and 8-10 are complex tasks. These tasks will be worked on by the developer agents (the Scrum team) with varying capabilities. The developer's capability ranges from 1 to 10 where 1 - 4 is considered novice, 5 – 7 intermediate and 8 - 10 expert. Ideally, a developer agent should only work on task with complexity equal to its capability. However, in situations where this is not possible, other options can be explored such as pairing with another agent or work solo on the task with penalty. It is also assumed that the team composition and the tasks are dynamics (these configurations vary in each sprint) and that certain strategy may work well for certain environment. These strategies are described in detail in the next sections.

3.1.1 Solo strategy

The solo strategy allocates a task to a single agent based on the capability of the available agents. The task is assigned to the agent with the closest capability with the complexity of the task. A preference

value (agent capability – task complexity) is used to determine which agents can undertake a task within the simulation (Wang, 2019a, 2019b). An agent is only permitted to work on tasks where the preference value is greater than or equal to -3. , This kind of setting is used to describe the gap between task complexity and developer's level of expertise. For example, if the gap between the task complexity and developer's level is lower than -3, this means the developer is not capable to work on the task. This also means in a real world setting, a novice developer is not able to work on a complex task that can only be worked by an expert. The time required to complete a given task is dependent on the agent's capability and the task's complexity. If the agent's capability is greater than or equal to the task complexity, then the time required to complete the task is equal to the task size (Equation 3.1).

$$\boldsymbol{TimeRequired = TaskSize} \qquad \text{Equation 3.1}$$

If the agent capability is lower than the task's complexity, the time required to complete a task is calculated as:

$$\boldsymbol{TimeRequired = |AgentCapability - TaskComplexity| * 0.33 * TaskSize + TaskSize}$$

Equation 3.2

Here, the time required to complete the task is longer since an agent with a lower capability will need more time to work on the given task. Since a gap of -3 is allowed, 0.33% is added to the time required to complete the task for each gap value (-1, -2 and -3) based on Griffith et al. (2014) where, for a given task that can be worked in a 1.0 unit of time by a middle level worker, it will take a 0.5 time unit for the senior to complete and 2.0 units of time for the junior to complete the task.

As discussed previously, a developer agent's capability can be categorised as novice, intermediate or expert agents such that the same level of pairings (expert-expert, intermediate-intermediate, and novice-novice) and cross level pairings (expert-intermediate, expert-novice, and intermediate-novice) are possible. However, based on the literature (Arisholm et al., 2007; Dybå et al., 2007; Lui & Chan, 2006; Nilsson, 2003) the effectiveness of the pairing is dependent on the level of expertise of the individuals in the pair and the task to be worked on. The following section provides an analysis of the different pairings.

3.1.2 Same level pairing

Same level pairing refers to pairing of developers with the same capability to work together on a specific task. This is the original idea for pair programming, which aimed to enhance the quality of code

cooperation between developers who enjoyed the work of coding to promote product delivery (Sommerville, 2016).

Expert-expert pair

This type of pair is considered as negative pairing as both experts may have disagreement on software design, coding and analysis. Dybå et al. (2007) supports two experts working on complex tasks as a good choice but suggests that experts should not work in pair when working on an easy task. Similarly, Arisholm et al. (2007) also supports two experts pairing to work on complex tasks but not on easy tasks. In addition, Lui & Chan (2006) shows that once a novice pair moved up to expert rank, it is no longer useful to pair them.

Intermediate-intermediate pair

This type of pairing is only effective when the pair is working on a complex task. Dybå et al. (2007) supports two intermediate experts working on complex tasks as a good choice, but a bad one when working on an easy task. Arisholm et al. (2007) also supported two intermediate pairs working on complex tasks but not on easy tasks. A single intermediate developer is not able to work on a complex task. However, pairing two intermediate developers can enhance the performance of each developer. This pairing is not recommended when working on intermediate tasks as an intermediate task can be worked on by a single intermediate expert (Arisholm et al., 2007; Dybå et al., 2007; Hannay et al., 2009; Lui & Chan, 2006).

Novice-novice pair

This type of pairing is not recommended when working on complex tasks as it is not possible for two novices to solve complex problems due to gaps in knowledge. A novice-novice pairing is effective when working on easy and intermediate tasks. Based on the work Dybå et al. (2007) and Arisholm et al. (2007) it is possible for two novices to work on all tasks , but two novice are not allowed to work on complex task, because this violates the rules that we defined in all strategies where agents are not allowed to work on a task with a gap greater than 3. If two novices are allowed to work on a complex task, this will result in significant time delay in delivering the task, and this is not a good strategy design, particularly for the Scrum team. In short, we should not use a strategy that result in task delivery delay as this will need longer completion time and higher person hours.

3.1.3 Cross level pairing

The purpose of cross level pairing is to facilitate learning and knowledge transfer between the pairs (Zieris & Prechelt, 2014). Good pairs can transfer knowledge, as supported by the work Plonka et al. (2015). Many researchers indicate that knowledge transfer between developers exists in an organization (Argote & Fahrenkopf, 2016; Lin & Zhang, 2019; Wang, Wang, & Zhang, 2019). By doing this, the lower level developer can learn from the high-level developer. Knowledge transfer occurs when pairings, such as when a student works together with a supervisor.

Expert-intermediate pair

This type of pair is effective as the intermediate can learn from the expert. This pairing is highly recommended when working on a complex task as this allows the intermediate to learn from the expert (Plonka et al., 2015). It is not recommended for this kind of pairing to work on easy and intermediate tasks as no learning can take place and the combined capabilities of the intermediate and the expert are not maximised.

Expert-novice pair

The expert-novice pair has the largest gap between the two developers, and this is considered an effective pairing as the novice can learn from the expert. A novice developer will not be able to work on an intermediate or complex task alone. However, by pairing with an expert, the novice has the opportunity to work on tasks at all levels.

Intermediate-novice pair

The intermediate-novice pair provides the opportunity for the novice to work on easy and intermediate tasks. The novice will also be able to learn from the intermediate. This pairing is not recommended when working on a complex task as the intermediate might not have the knowledge to tackle the task based on our strategy design.

Table 3.1 summarises the different pairings for the varying task complexities used in this simulation. Positive pairing is the preferred pairing as it results in better code quality and allows junior developers to learn from senior developers as we made the assumption that junior developer can learn from the senior developers (Lin & Zhang, 2019; Wang et al., 2019).

Table 3.1 Pair programming modelling assumptions

Pair	**Easy task**	**Intermediate task**	**Complex task**
Novice-novice	Positive	Positive	Negative (not used)
Intermediate-intermediate	Negative	Negative	Positive
Expert-expert	Negative	Negative	Negative
Novice-expert	Positive	Positive	Positive
Intermediate-expert	Negative	Negative	Positive
Intermediate-novice	Positive	Positive	Negative (not used)

There are several situations where pairing is not recommended (negative pairing):

- Novice-novice working on a complex task
- Expert-expert (all cases)
- Intermediate-intermediate working on intermediate and easy tasks
- Expert-intermediate working on intermediate and easy tasks
- Intermediate-novice working on a complex task

In our simulation, the time required to complete a task worked on by a negative pair will be inflated by 15% to accommodate the overheads (misunderstandings and disagreements) from this pairing. (Arisholm et al., 2007).

When pairing, the pair capability is based on the capability of the stronger agent (referred to as the lead agent). The time required to complete a task for a pair is half the time taken by a solo developer as there are two developers working on the same task. For negative pairing, a penalty is imposed by increasing the time required to complete the task by the pair by 15% (Arisholm et al., 2007). Negative pairing may also affect the quality of the completed task. In our modelling we only considered the time penalties required, as the negative pair can result in low team performance that reduces the team's efficiency. The time required to complete a given task is as follows:

$$TimeRequired = \begin{cases} TS, LAC \geq TC \\ (|LAC - TC| * 0.33 * TS) + TS, LAC < TC \end{cases} \quad \textbf{Equation 3.3}$$

where LAC is the lead agent capability, TS is the task size and TC is the task complexity. As before, 0.33 is the additional time imposed by a developer working on a task that has a complexity higher than the developer's capability (0.33 for a gap of 1, 0.66 for a gap of 2 and 1.0 for a gap of 3). In this scenario,

the time required to work is longer since the agent with the lower capability needs more time to work on the given task.

Here, the time required to complete the task is longer since an agent with a lower capability will need more time to work on the given task. Since a gap of -3 is allowed, 0.33% is added to the time required to complete the task for each gap value (-1, -2 and -3) based on Griffith et al. (2014) where, for a given task that can be worked in a 1.0 unit of time by a middle level worker, it will take a 0.5 time unit for the senior to complete and 2.0 units of time for the junior to complete the task.

The time required is updated based on the nature of the pairing where negative pairing has a penalty imposed as follows:

$$TimeRequired = \begin{cases} TimeRequired, if\ no\ penalty \\ 1.15 * TimeRequired, with\ penalty \end{cases} \quad \textbf{Equation 3.4}$$

The time required for each individual agent in the pair is:

$$TimeRequired_A = \frac{TimeRequired}{2} \quad \textbf{Equation 3.5}$$

as each agent can split the work equally. In Lui & Chan (2006) they indicated that pairs will use half the time to complete the same task compared with a solo.

3.1.4 Strategy types

In our simulation, we modelled our pairing based on the following:

- Type 1 – the lead agent must have a capability equal to or higher than the task. In this setup, the lead agent can choose who to pair with depending on the task complexity.
- Type 2 – the lead agent must be in the same category or higher than the task. This means that a novice agent can work on any easy task, an intermediate agent can work on medium and easy tasks, and an expert agent can work on complex, medium and easy tasks.
- Type 3 – similar to type 2, but the lead agent with a gap (agent capability – task complexity) of -3 or greater can work on the task. By allowing a gap of three, it is possible for two intermediate agents to work on a complex task (Dybå et al., 2007). Also, in this setup, a pair of novice agents can work on a medium task.

These strategies were set up to observe how they affect the team's efficiency, completion time and idle time. In Type 1, the pair can only work on a task equal to or lower than the pair's capability. The

Type 2 strategy allows the pair to work on a task in the same category as the capability of the pair, while Type 3, allows tasks in a higher category to be worked on by the pair (e.g. an intermediate task can be worked on by a novice pair and a complex task can be worked on by an intermediate pair).

3.1.5 Must pair strategy

In the must pair strategy, each task must be undertaken by a pair of agents as no agent can work on the task alone. In this strategy, each task needs to be verified by a pair of agents who have not worked on the task previously. To accommodate the must pair, a single agent must wait for another agent to pair with before working on another task. This also means that some pairings may be ineffective. The must pair strategy has three sub-types as shown in Tables 3.2 and 3.3.

Must pair Type 1 (MP1)

Table 3.2 Must pair Type 1

Task	First choice	Second choice	Third choice	Fourth choice
Easy	Novice	Novice	Intermediate	Expert
	Novice	Any agent	Any agent	Any agent
Intermediate	Intermediate	Intermediate	Expert	Expert
	Novice	Any agent	Novice	Any agent
Complex	Expert	Expert	Expert	
	Novice	Intermediate	Expert	

For easy tasks in this strategy, a novice who has a capability equal to or higher than the task complexity is the lead agent. In this situation the novice agent should pair with another novice agent. If no novice agent is found, then it may pair with any agent. However, if no novice agent is available to work on the easy task, an intermediate agent can be nominated as the lead agent and may pair with any other agent. Finally, when there are no novice or intermediate agents available, an expert, takes the role of a lead agent and can pair with any other agent.

For an intermediate task, an intermediate agent who fulfils the conditions will be the lead agent and must first pair with a novice (to provide an opportunity for the novice to learn), followed by any agent. If an intermediate agent is not available, the expert agent will be the lead agent who will then pair with a novice first (if available), followed by any agent (Arisholm et al., 2007).

For a complex task, an expert agent with the right capability is the lead agent and may first pair with a novice, followed by an intermediate and an expert (Arisholm et al., 2007).

Must Pair Type 2 (MP2)

The MP2 strategy is similar to MP1, so the pairing decisions are as in Table 3.2. However, the main difference in this strategy is that we allowed some flexibility on what task the agents can work on. A pair of agent can work on a task as long as the task complexity is in the same category as the lead agent's capability (i.e. an easy task can be worked on by a novice agent, a medium task can be worked on by any intermediate agent, and a complex task can be worked on by any expert agent).

Must Pair Type 3 (MP3)

Must Pair Type 3 is decided to fit with the type 3 strategy required, which allowed lower capable agent to work on higher complexity task within -3 gap.

Table 3.3 Must pair Type 3

Task	First choice	Second choice	Third choice	Forth choice	Fifth choice
Easy	Novice	Novice	Intermediate	Expert	
	Novice	Any agent	Any agent	Any agent	
Intermediate	Intermediate	Intermediate	Expert	Expert	Novice P-value>-4
	Novice	Any agent	Novice	Any agent	Novice
Complex	Expert	Expert	Expert	Intermediate P-value>-4	
	Novice	Intermediate	Expert	Intermediate	

MP3 is another variation of MP1 where, for this strategy, a novice-novice pair is allowed to work on an intermediate task and an intermediate-intermediate pair is allowed to work on a complex task.
This strategy allowed the intermediate and novice to work on higher level tasks and this is very useful when the team's maximum capability is lower than the task required.

3.1.6 Intelligent pair strategy

The intelligent pair strategy (voluntary pair) ensures that agents only pair if the pairing is positive and, in situations where pairing is not possible, the task will be assigned to a solo agent with sufficient capability. Each task needs to be verified by an agent who has not worked on the task (Schwaber & Sutherland, 2017). The intelligent pair strategy has three sub-types that shows in Tables 3.4 and 3.5.

Table 3.4 Intelligent pair Type 1

Task	First choice	Second choice	Third choice	Fourth choice
Easy	Novice	Novice solo	Intermediate solo	Expert solo
	Novice			
Intermediate	Intermediate	Intermediate solo	Expert	Expert solo
	Novice		Novice	
Complex	Expert	Expert	Expert solo	
	Novice	Intermediate		

In this strategy (see Table 3.4), effective pairing has a higher priority. For an easy task, if there is a novice lead agent available, it will try to pair with another novice, followed by an intermediate and then an expert. If there are no agents available to pair with at that time tick, the novice agent will work on the task independently. If there is no novice agent, then a solo intermediate agent is preferred over an expert agent.

For an intermediate task, the intermediate agent is the lead agent and will pair first with a novice agent. If this pairing is not possible, a solo intermediate is the next choice, followed by an expert-novice pair and then an expert solo.

For a complex task, the expert agent is always the lead agent with the novice agent as the pair, followed by the intermediate. If novice and intermediate agents are not available, an expert agent will work on the task individually.

Intelligent pair type 2 (IP2)

The IP2 strategy is similar to IP1, so the pairing decision is the same as in Table 3.4. As before, the main difference is that in this strategy, we allow some flexibility on what task the agents can work on. An agent can work on a task as long as the task complexity is in the same category as the agent's capability (i.e. easy tasks can be worked on by a novice agent, a medium task can be worked on by any intermediate agent, and a complex task can be worked on by any expert agent).

Intelligent pair Type 3 (IP3)

Table 3.5 Intelligent pair Type 3

Task	First choice	Second choice	Third choice	Fourth choice	Fifth choice	Sixth choice
Easy	Novice	Novice solo	Intermediate solo	Expert solo		
	Novice					
Intermediate	Intermediate	Intermediate solo	Expert	Expert solo	Novice P value >-4	Novice solo P value>-4
	Novice		Novice		Novice	
Complex	Expert	Expert	Expert solo	Intermediate P value>-4	Intermediate solo P value>-4	
	Novice	intermediate		Intermediate		

IP3 is another variation of IP1 where in this strategy, a novice-novice pair and a novice solo are allowed to work on an intermediate task and an intermediate-intermediate pair and an intermediate solo pair are allowed to work on a complex task. This strategy is similar to MP3 in that it allows intermediates and novices to work on higher level tasks.

The IP3 is designed particular for testing against IP1 and IP2, to see how lower capable team behave when working on high complex tasks, which IP1 and IP2 are not allowed to do. This extends the range of the strategy application scale, which maximise the utility of team resource to work in as many task as possible, to avoid idle time.

3.2 Evaluation Metrics

To evaluate the performance of the various strategies, five different metrics are defined:

Completion time

This is the elapsed time used by the team to complete all the tasks (for the pre-defined user stories) from the start of the sprint until the end of the sprint when all tasks in the sprint backlog were completed, verified and marked as completed.

The completion time is originally from the idea of elapsed time, which is used to indicate if the project can be completed on time or not in software engineering domain. This research also uses this metric to indicate the length of project, which is useful to see which strategy use the shortest completion time.

Person hours

$$PH = CT \times NA \qquad \textbf{Equation 3.6}$$

The person hours can be used to calculate the cost for the team to complete the product backlog. It has an economic purpose. This is calculated by multiplying the completion time with the number of agents in the team as shown in Equation 3.6 where PH is person hours, CT is completion time and NA is the number of agents in the team.

The person hours used to estimate the time accumulated by the whole team. The longer the elapsed time and team numbers, the higher the person hours, which result in higher project cost. This research will use this metric to estimate the cost needed by various strategies.

Idle time

The idle time indicates the amount of time that an agent was available but not actively working on a task. The total idle time is collected for a Scrum team in each run.

The idle time is to estimate the whole team's idle time in which the idle time is created either because no task can be allocated to the developer or there is a constrains of task allocation on the developer. Each strategy would record different idle time, and so we are interested to observe the difference in idle time where the lower the idle time the better the performance.

Effort time

This indicates the amount of time an agent was working and busy. The total effort time is collected for a Scrum team in each run.

The effort time is indicates the work completed by each developer. Higher effort time indicates that the team is busy. Different strategies will have different effort time, and so it is useful to observe the performance of various strategies with respect to effort time.

Working efficiency

The working efficiency is the teamwork efficiency. This is obtained by dividing the completion time with the average workload of the agents (the lower the better). Here, working efficiency (WE) is defined as:

$$\mathbf{WE} = \frac{\mathbf{CT}}{\left(\frac{\mathbf{WL}}{\mathbf{NA}}\right)} \qquad \textbf{Equation 3.7}$$

where CT is the completion time, WL is the total workload (summation of all the tasks in the sprint backlog) and NA is the number of agents in the team.

The working efficiency determine the work per person per workload to complete a project.

3.3 Experimental Setup

The purpose of this experiment is to investigate the performance of strategies under various settings namely fixed environment and random environment. In the fixed environment, the number of team is set to five or six agents with varying capabilities such as a novice dominated team, an intermediate dominated team, an expert dominated team and an evenly distributed team. This number is taken as the midpoint as the Scrum guide recommends a team to have between four and nine members (Schwaber & Sutherland, 2017). Each sprint comprised 50 tasks (10 user stories and five tasks for each story), with varying complexities. This is useful because in a real world setting, each user story can be composed of different level of tasks. The team can decide themselves the size of each task in a user story based on this relative estimation. The tasks to be worked by the team in each sprint can be classified as many easy tasks, many intermediate tasks, many expert tasks and evenly distributed tasks. In the random environment, the number of agents, the agents' capabilities and the task complexities are generated randomly to mimic the real-world setting. We are also interested in investigating what happen when there are potential conflicts in the pair within the team. Here, only one of the strategies is tested.

In these experiments, these strategies are executed based on a single sprint. Hence, the last experiment investigates how the selected strategy can be implemented in a real world setting by running multiple sprints until all the user stories in the product backlog are completed using varying sprint durations. The details of these experiments are discussed in the next sections.

3.3.1 Fixed testing

In the fixed environment, we fixed the number of agents in a team between five and six agents. There are eight types of team setup: five agents with many novices; five agents with many intermediates; five agents with many experts; five agents with an evenly distributed capability; six agents with many novices; six agents with many intermediates; six agents with many experts; and six agents with an evenly distributed capability.

These type of teams are simulated as suggested by the scrum guide, where it is recommended have around five to six people to compose a team. To simulate team composition, we then designed the above team distribution with novice, intermediate, expert and evenly distributed. This is very useful

to investigate how different team composition can affect the scrum strategy performance, so that we can answer the research questions.

The setup for these eight types of team is enough for us to observe the performance of these seven strategies and to understand the effect the team composition may have on these strategies. Table 3.6 shows the team compositions for the eight teams and the agents' capabilities in each team.

Table 3.6 Team for fixed testing

Agent Capability	Five Developer team (many novices)	Five Developer team (many Intermediates)	Five Developer team (many experts)	Five Developer team (even distribution)	Six Developer team (many novices)	Six Developer Team (many intermediates)	Six Developer team (many experts)	Six Developer team (evenly)
1	one			one	one			one
2	one			one	one			one
3	one				one			
4		one	one			one	one	
5	one	one			one	one		
6		one		one	one	one	one	one
7		one	one	one		two	one	one
8	one	one	one	one	one	one	one	two
9			one				one	
10			one				one	

The tasks are distributed based six scenario shown in Table 3.7. The Type 1 scenario set has 10 easy tasks, 20 intermediate tasks and 20 complex tasks. The Type 2 scenario has 20 easy tasks, 10 intermediate tasks and 20 complex tasks. The Type 3 scenario has 20 easy tasks, 20 intermediate tasks and 10 complex tasks, the Type 4 scenario has 30 easy tasks, 10 intermediate tasks and 10 complex tasks, the Type 5 scenario has 10 easy tasks, 30 intermediate tasks and 10 complex tasks. The Type 6 scenario has 10 easy tasks, 10 intermediate tasks and 30 complex tasks. Scenario 1, 2, and 3 are variations of the evenly distributed tasks, while Scenario 4 has more easy tasks, Scenario 5 has more intermediate tasks and Scenario 6 has more complex tasks. We set up these various task sets in order to see how the teams reacted to those different task sets based on the use of different strategies.

The above design is set up so that we can observe how different project setup with different level of complexities can affect the scrum team strategies performance.

Table 3.7 Task set categories

Scenario	Description
One	Ten easy tasks, 20 intermediate tasks, 20 difficult tasks. Each user story with 30 as the workload. Total of 50 tasks.
Two	Twenty easy tasks, 10 intermediate tasks, 20 difficult tasks. Each user story with 28 as the workload. Total of 50 tasks.
Three	Twenty easy tasks, 20 intermediate tasks, 10 difficult tasks. Each user story with 26 as the workload. Total of 50 tasks.
Four	Thirty easy tasks, 10 intermediate tasks, 10 difficult tasks. Each user story with 25 as the workload. Total of 50 tasks.
Five	Ten easy tasks, 30 intermediate tasks, 10 difficult tasks. Each user story with 30 as the workload. Total of 50 tasks.
Six	Ten easy tasks, 10 intermediate tasks, 30 difficult tasks. Each user story with 34 as the workload. Total of 50 tasks.

Each testing is made up of 10 user stories where each user story consists of five tasks with varying levels of complexity. The distribution of the task complexities for each user story are shown in Table 3.8.

The reason why each user story is composed of 5 tasks is because this research only considers task allocation, and because scrum team can only break down each user story and work on the task directly. We thought there are 5 to 6 developers in the team, so that during the user story break down, they would prefer to break each user story into 5 tasks to work on it as the average number.

It is also because the design of 5 tasks to a user story is an appropriate size, such that the smallest user story is 5 and the largest user story is 50. Such size variation guarantees that a small user story can be delivered within one day, and the largest user story can be delivered in two weeks. If the user story is too large, then it mayoccupy too much time which may take longer than two weeks to complete.

In our modelling, we regard one day is equal to 5 time ticks, and a week is equal to 25 time ticks, two weeks equal to 50 time ticks, three weeks equal to 75 time ticks and four weeks equal to 100 time ticks. Based on these assumptions, we use 5 tasks to compose a user story, which ensures that a user story can be completed between one day and two weeks, depending on its size, which is appropriate and compatible with the real world setting.

However, as the scrum team is fully responsible for its user story definition, the team can decide on the size of the user story.

Table 3.8 User stories for each test case

Task Complexity in each user story	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
1						
2	one	one	one			
3		one	one	one		
4				two	one	one
5	one		one		one	one
6	one	one		one	one	
7			one		one	
8	one	one		one	one	two
9	one	one	one			one
10						

All possible combinations

There are 48 environments in total as shown in Table 3.9. In each environment, the seven strategies are tested to see how they performed and based on that, we can analyse how each strategy performed under each environment. Each row shows the number of agents with their capabilities and the different scenarios used to test all seven strategies. More novices indicate that the majority of the agents were novices (e.g. for five agents the breakdown is 3:1:1 which meant three novices, one intermediate and one expert). For each environment, we run the sprint repeatedly (10 times) and observed the completion times, average effort time, average idle time, and the work efficiency for each strategy.

Table 3.9 All possible combinations

Team Composition	Test case 1 (10:20:20) Complex	Test case 2 (20:10:20) complex	Test case 3 (20:20:10) intermediate	Test case 4 (30:10:10) easy	Test case 5 (10:30:10) intermediate	Test case 6 (10:10:30) complex
Five agents More Novice 3:1:1	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3
Five agents more intermediate 1:3:1	SOLO?? MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3
Five agents more experts 1:1:3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3
Five agents evenly distributed 2:2:1	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3
Six agents more novices 3:2:1	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3
Six agents more intermediates 1:4:1	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3
Six agent more experts 1:2:3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3
Six agents evenly distributed 2:2:2	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3	SOLO MP1 IP1 MP2 IP2 MP3 IP3

3.3.2 Random testing setup

The purpose of random testing is to simulate up a real-world situation where we could not predict the team composition or what type of task set the team is going to work on. We also wanted to observe which strategy worked best in this random situation. These random tests were repeatedly run, up to 100 times, to explore the various combinations of team and task. In each run the number of agents generated is between five and six.

In the real word, the team and task distribution are variable. This random setup, allows us to simulate this situation.

We then observed how these strategies performed under unknown situations. The task complexity is randomly generated between 1 and 10, similarly, the agent capability is also randomly generated between 1 and 10. This is repeated 100 times for all strategies. Figures 3.1 and 3.2 show the distribution for the tasks and agents for 100 runs. We also introduced an additional strategy (referred

to as adaptive strategy) where the strategy used is dependent on the environment it is in. For example, if the current environment has 6 agents in a team that needs to work on many complex tasks, then the strategy to be used is the strategy that performed well in that fixed environment. In every run, the strategy to be used is changed accordingly.

Figure 3.1 shows the accumulated tasks in the random test and its distribution. Figure 3.2 shows the accumulated agents in the random test and its distribution. The 100 runs are sufficient because the data distribution for 10 run, 20 runs, 30 runs, 100 runs, 500 runs are found to have similar the team capability and task complexity. This is the same for the task complexity.

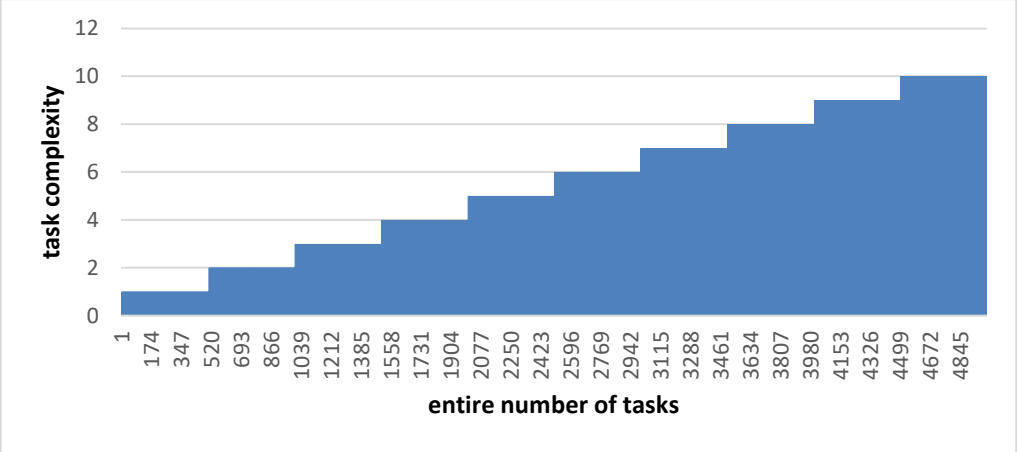


Figure 3.1 Task complexity distribution

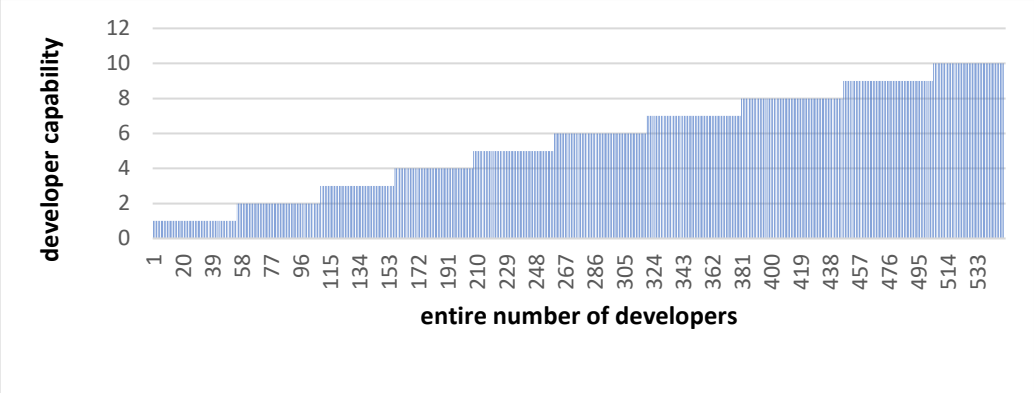


Figure 3.2 Agent capability distribution

3.3.3 Modelling team conflicts

Team conflict is modelled because we assumed there could be conflicts in the pairs where both members are not in agreement when completing the allocated tasks.

In the fixed and random environment, we assumed that there is no conflcit within a pair. However, in the real world, conflict often happens between working partners and so we intentionally model the various conflict rates in this setting to see how it can affect the IP3 strategy's performance.

This conflict may result in additional work where 30% of the task would need to be reworked. The conflict rates were set to 5%, 10%, 15% and 20%. In this experiment, we use IP3 strategy only as this strategy allows most of the developers with even low capability to work on complex tasks.

To reflect how this strategy can be implemented in a real world setting, it is useful to investigate how these strategies can be implemented in a software development project. In the previous experiments, all the strategies were run in a single time sprint without setting a maximum duration of each sprint. In this experiment, we investigate the effect of implementing IP3 using varying sprint durations. Here, we use 25 to represent a week sprint, 50 to represent two weeks sprint, 75 to represent 3 weeks sprint and finally 100 to represent 4 weeks sprint. In this experiment, we use IP3 as the team strategy as before.

The multi-sprint works in a similar fashion with the single sprint, but it is repeated multiple times until all the user stories in the product backlog are completed. Each sprint is setup to a specific time box and tasks can only be allocated if that task can be completed within the current sprint. If there is still time remaining, the task that can be completed in the current sprint can be searched from the product backlog. We then observed the performance of IP3 in terms of person hours, completion time, effort time, idle time, and efficiency.

3.4 Linkage to the Research Questions

This section explains how Chapter 3 is linked to the research questions, including how various strategies design are linked to the research questions as shown in Table 3.10 and as well as how the performance metrics are linked to the research questions and is shown in Table 3.11.

Table 3.10 Strategy and linkage to the research questions

	Type 1	Type 2	Type 3	Purpose	Research questions addressed
Solo				Comparing solo and pair programming	RQ1, RQ2, RQ3, RQ4
Must pair	MP1	MP2	MP3	Pure pair programming to versus variants of pair programming	RQ1, RQ2, RQ3, RQ4
Intelligent pair	IP1	IP2	IP3	Comparison of voluntary pairing with must pair and solo	RQ1, RQ2, RQ3, RQ4

The performance metrics used in these comparisons in varying environments are shown in Table 3.11.

Table 3.11 Performance metrics in varying environments

	Average completion time	Average Person hours	Average effort time	Average idle time	Average efficiency value	Project completed (True/False)
Fixed testing	yes	yes	yes	yes	yes	yes
Random testing	yes	yes	yes	yes	yes	yes
Conflict	yes	yes	yes	yes	yes	yes
Multi-sprint	yes	yes	yes	yes	yes	yes

Research question and its linkage between methods and experiment.

1. How does the adoption of pair programming impact the effort required to implement a software project?

This question will be answered by evaluating the performance of solo, must pair and intelligent pair strategies in the fixed environment based on average completion time, average person hour, average effort time, average idle time, average efficiency value and whether the sprint can be completed or not.

2. What impact does the chosen pairing scheme have on the effort required?

This question will be answered by analysing the performance of solo, must pair and intelligent pair strategies in the fixed environment and random environment. Analysis performed include investigating the impact of cross-level pairing, same level pairing, must pair and voluntary pairing based on those metrics shown in Table 3.11

3. How does team composition impact project completion under different pairing schemes?

This question is answered by exploration of the different team compositions (defined by the strategies) and analysing their effect on the project completion.

4. In which situations is pairing advantageous compared with not pairing?

In order to answer this question, several situations are considered, such as random context, conflict context and multi-sprint context.

The experiments and its linkage to the research questions are detailed in Table 3.12.

Table 3.12 Experiments and linkage to research questions

Environment	Research questions	Purpose
Fixed 48 context (8 types of team and 6 sets of tasks)	RQ1, RQ2, RQ3, RQ4	To observe the performance of solo versus must pair and intelligent pair. To compare cross level pairing, same level pairing by observing the effort. To observe the project completion. To observe the benefits of pairing.
Random task complexity and agent capability	RQ4	Simulate a real-world situation to observe which strategy worked best in this random situation.
Conflict testing on IP3	RQ4	To model conflicts in pairs.
Multi-sprint testing on IP3	RQ4	To model multi-sprint setting.

3.5 Summary

This chapter describes six strategies that can be used when implementing pairing schemes in a Scrum software development project. The performances of these strategies are compared with the performance of strategy that implements solo programming. The summary of the strategy design features is shown in Table 3.13.

Table 3.13 Strategy design feature comparisons

Strategy	Use same level pair	Use cross level pair	Use negative pair	Use positive pair	Use solo
Must pair	yes	yes	yes	yes	No
Intelligent pair	yes	yes	no	yes	yes
Solo	no	no	no	no	yes

Based Table 3.13, we expect the intelligent pair to perform well. This is because the intelligent strategy will only pair if the pairing is positive and when no positive pairing is possible, it will resort to working solo. This is not the case for must pair, where positive pairing is desired but when this is not possible the strategy still forces the pair to work on the task. We will test these strategies and verify it in the experiments.

In addition, the performance measures for these strategies are described in detail. These strategies are to be tested in varying environment (fixed and random) to observe its performance and the effect of pairing in varying situations. Further, two additional experiment are setup to investigate the effect of pairing conflicts within a team and the effect of varying sprint durations in a multi-sprint setting. The results of these experiments will be presented and discussed in the Chapter 5. The next chapter discusses the design and implementation of the MAS Scrum simulation.

Chapter 4

Design and Implementation of the Scrum Simulation

This chapter sets out the design and implementation of an agent-based model to simulate and evaluate strategies for the Scrum software development process. Agent based modelling (ABM) is a powerful simulation modelling technique that allows the modeller to simulate real-world problems (Jennings & Wooldridge, 1998; Nicholas & Michael, 1998; Wooldridge & Jennings, 1995). In ABM, a system is modelled as a collection of agents who can make autonomous decisions. In this project, the ABM model is implemented using a multi-agent system (MAS) to allow the agents to make intelligent decisions based on specific rules as well as to capture both the macro and micro levels of the interactions between the agents and the environment. As discussed in Chapter 2, in a Scrum environment, there are several key players: namely, the development team, the Scrum Master and the Product Owner. When using MAS, different agents can be created to mimic the behaviour of the different roles in Scrum. This chapter describes an ABM, agent and its characteristics, the multi-agent design of the simulated environment and the design of the individual agents. In addition, this chapter also discusses how a set of prioritised user stories are broken into tasks and worked on by the developer's team in individual sprint iterations until they are completed.

4.1 Agent-based Modelling

The main idea of ABM is that many real-world phenomena can be effectively modelled as an agent, an environment and as a description of agent-agent interactions and agent-environment interactions (William & Uri, 2015). Modelling the real world involves developing a computational model that comprises agents and the environment in which the main actors are the agents (Gilbert & Troitzsch, 2005; Mehmood, Ahmed, & Kristensen, 2019)

Agent-based modelling can be used to model social behaviour and interactions. Boulahbel-Bachari and El Saadi (2019) used agent-based modelling to model the behaviour of migration and self-selection of people. Joslin & Poole (2005) and Macal & North (2006) provided an overview of how to use agents to model software project planning by modelling the developer team. In this work, agents are used to model and simulate human developers, such that an agent can have capability, skills and experience. Agents can pick up a task and work on it and deliver the product. Different agents can have different skill sets and motivation. Task modelling is important when constructing the scale of the overall project. Agents' interactions with an agent or a task can be captured using agent-based modelling, and such modelling can model the process of how each agent completes the task.

Agent-based modelling is suitable for undertaking team-based research to capture team behaviour and interaction when carrying out software project estimations and optimization (Agarwal, 2007; Agarwal & Umphress, 2010; Macal & North, 2008; Phillips, 2006; Yilmaz & Phillips, 2007). Team-based modelling uses multi-agents, which are a group of agents that interact with each other in a team. The shared goal of these agents is to complete all the tasks. In the real world, a software developer team usually contains more than one developer, and an agent can be used to model and simulate each developer and to group them as a team. The reasons for this are based on an agent's unique characteristics, especially its features of human behaviour and decision-making. Agent-based modelling can be used to model a group of agents and verify their impacts on team goal realization and achievement. Such features can be used to undertake team-based analysis in particular organizational-based contexts.

Abar, Theodoropoulos, Lemarinier, & O'Hare (2017) specifically reviewed all the tools that have been developed for agent-based modelling and simulation, classifying them by their functionality and scales of computing that can be applied. These tools can be distributed for various modelling purposes, such as human behaviour modelling, animal behaviour modelling and cell modelling in biology. One of the more popular tools is NetLogo (William & Uri, 2015), which is used to simulate natural and social phenomena and is well suited for modelling complex systems over time. NetLogo allows developers to explore the various agents' behaviour under a variety of settings.

For the purpose of this study, a multi-agent system was used to simulate the Scrum environment to enable agents to make autonomous decisions and interact with one another using a standard agent communication language. The next sections describe the agent and the multi-agent systems in some detail.

4.1.1 What is an agent and its characteristics?

Agent is a special software component that can make decisions autonomously based on its inner rules and logic reasoning (Wooldridge & Jennings, 1995). An agent can be defined as a computer system that is situated in some environment that is capable of autonomous actions in order to meet its design objectives. Agents can be used to model human behaviour or to construct an intelligent system (Wooldridge & Jennings, 1995).

Wooldridge and Jennings (1995) noted that computer agents typically have the following properties:

• Autonomy – agents operate without having direct control over their actions and internal state.

• Social ability – agents interact with other agents through some kind of 'language' (a computer language, rather than a natural language);

• Reactivity – agents are able to perceive their environment (which may be the physical world, a virtual world of electronic networks or a simulated world that includes other agents) and respond to it; and

• Proactivity –agents are also able to take the initiative and engage in goal-directed behaviour.

Agents can communicate with each other through a common language structure, such as Foundation for Intelligent Physical Agent – Agent Communication Language (FIPA ACL), which is an agent language protocol standard, to exchange information (Bellifemine, Caire, & Greenwood, 2007).

4.1.2 What is a multi-agent system?

As mentioned previously, agents are special software than can act autonomously on behalf of the users to achieve specific goals. Agents often need to work together to solve more complex problems. A multi-agent system (MAS) is a loosely coupled network of agents that interact to solve problems that a single agent will not be able to solve alone (Ali et al., 2019; Tyrychtr et al., 2019). Agents may need to cooperate, coordinate, collaborate and negotiate with each other to achieve their goals in a MAS environment. The ability to interact is made possible by the presence of a standard agent communication language (ACL), such as FIPA-ACL (Bellifemine et al., 2007). Various MAS applications have been developed to solve a variety of problems in multiple domains, such as e-commerce (Tyrychtr et al., 2019), the military (Han, Liu, & Lei, 2019), networking (Boulahbel-Bachari & El Saadi, 2019), logistics and supply chains (Ivanov, Kapustyan, Kalyaev, & Korovin, 2019) and disaster rescue (Hashimoto et al., 2019) .

MAS is ideal for this research as each individual agent can be programmed to possess characteristics that embody the different roles in Scrum, such as the Scrum Master and developers. Each individual agent is able to make its own decisions about whether to take on a specific task, depending on its capability, as well as being able to make decisions about which agent to pair to gain an additional advantage. These developer agents can communicate with each other and the Scrum Master agent. They can also share information with one another either by means of communication or a shared blackboard. The Scrum MAS environment is described in the next section.

4.2 Scrum Environment, Agents and User Stories

Figure 4.1 shows an overview of the Scrum simulation using a multi-agent system. In this simulation, there are two types of agents, the Scrum Master agent and the developer agents. Following the Scrum guidelines, there are between four and eight developer agents and one Scrum Master agent (Schwaber

& Sutherland, 2017). This team of agents work through sprints to complete all the user stories in the product backlog. In each sprint, the agents' objectives are to work collaboratively to complete the user stories (which are broken into smaller tasks) in the sprint backlog. At the beginning of each sprint, user stories from the product backlog are moved into the sprint backlog. This will be repeated until all user stories in the product backlog are completed, at which time, the software development project will be deemed to be completed. Agents share and exchange information through the use of the Scrum board where agents can view information about the status of another agent, the task and the user stories.

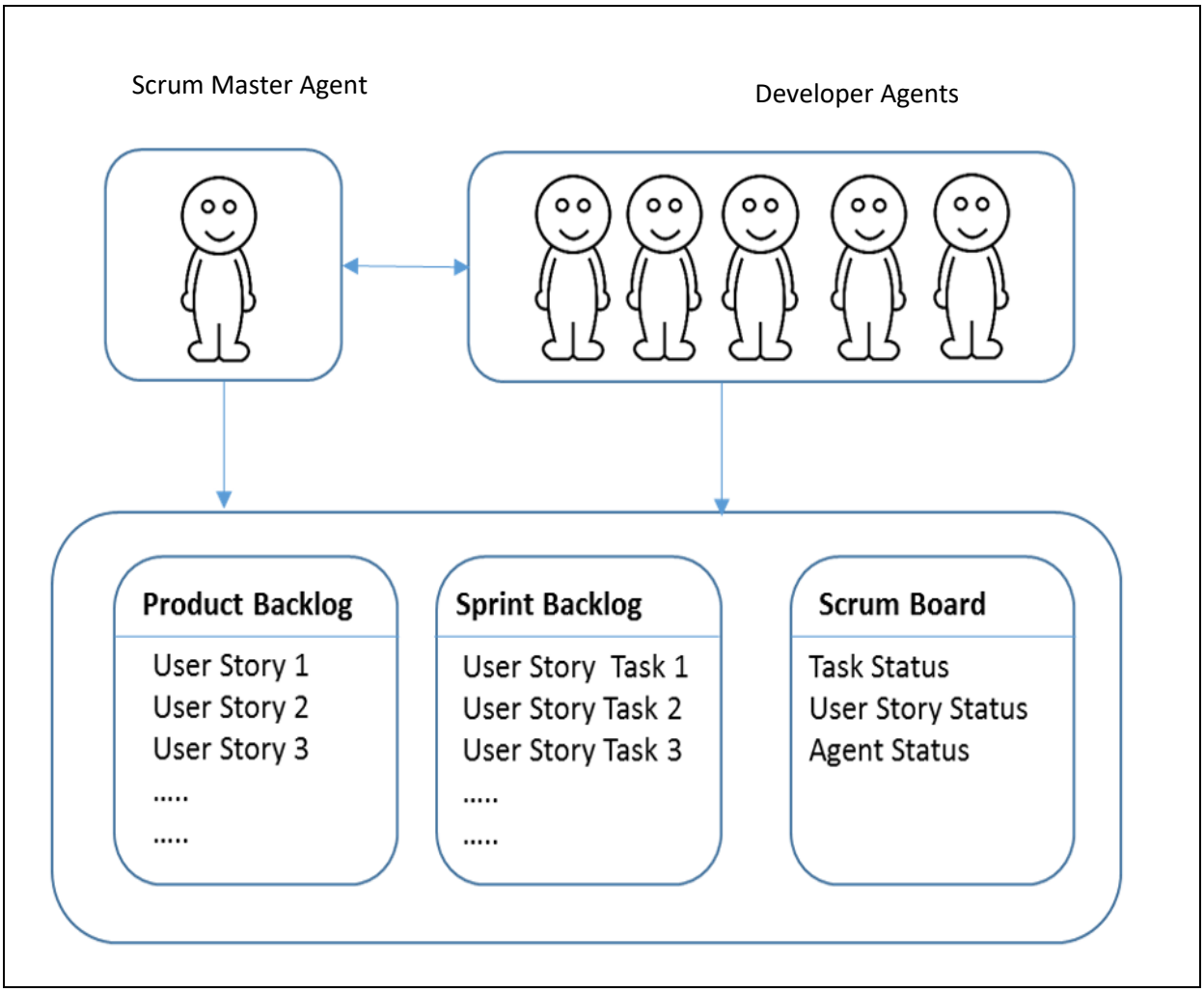


Figure 4.1 Overview of Scrum simulation

4.2.1 Scrum Master agent

The Scrum Master agent is designed to communicate with the developer agents during task allocation, to allocate tasks and to manage the Scrum board. The Scrum Master agent communicates with all developer agents to check their status, get their preferences for particular tasks, allocate tasks to each agent, form a pair based on the task requirements and the capability of the available agents, and

allocate tasks to the pairs. The number of agents and the number of user stories can be defined in the simulation. At the start of the simulation, agents (with different capabilities) and tasks (with different levels of complexities and sizes) are generated.

At the beginning of the sprint, the Scrum Master will coordinate the task allocations for the developer agents by sending a list of tasks that are available. Each agent sends its preferences to the Scrum Master and the Scrum Master decides the task allocations based on the agents' preferences[1]. At each time tick thereafter, each developer agents sends its status to the Scrum Master and in turn the Scrum Master updates the status of each agent and task. When an agent completes a task and becomes available, the Scrum Master allocates a new task to the agent, which the agent may accept or not depending on the task complexity and the agent's capability. This is repeated until all the tasks in the sprint backlog are cleared and marked as completed.

A task can be allocated to a single agent (referred to as a solo programming) or it can be allocated to a pair of agents (referred to as pair programming). For solo programming, the Scrum Master allocates the tasks to agents based on task complexity and agent preference, whereas, to allocate the task to a pair it chooses the two most appropriate agents to work on the task based on a pairing strategy.

The Scrum Master also initiates the verification of the task once it is completed by adding a verification task in the sprint backlog. When a user story is completed (all the tasks for that user stories are completed and verified), the Scrum Master initiates a user story verification task, which also gets added to the sprint backlog. The Scrum Master also manages the Scrum board where it updates all the information relating to the sprint, such as the agent's status, the agent's idle time, the agent's working time, task status, task start time, task end time, task verification time, user story status, the current time tick and the sprint completion time.

4.2.2 Developer agents

The main role of the developer agents is to work on completing the tasks assigned to them. They communicate with the Scrum Master to provide information about their status, the status of the task they are working on and their task preference. Each developer agent has a name, capability and status. The capability represents technical capability ranging from 1 to 10 where 1-4 is considered novice, 5 – 7 intermediate and 8-10 expert[2]. This capability value is then used to determine how much time is required by the individual agent to complete a given task. Each agent has a status to indicate their

[1] This is a slight variation from the Scrum setting where developers (including the Scrum Master) negotiate on which tasks they want to work on. In this implementation, we have made Scrum Master a coordinator agent to reduce communication threads between all the agents. However, all tasks are allocated based on the agents' preferences.

[2] In the real world, a developer may have specific capabilities in programming, design, and database. In this work, a single value is used for simplicity.

current activity (working, verifying, idle). Working indicates that the agent is currently working on a task, verification indicates that the agent is verifying a task that has been completed by another agent and idle indicates that the agent is free and available to take on work.

4.2.3 User stories and tasks

Each user story is composed of multiple tasks. In this simulation, each user story is assumed to have five tasks. This is based on data collected from a number of software companies in Christchurch at the beginning of this study in 2017. Each task has a size, which indicates how much effort is needed to complete the task (the more complex the task, the larger its size). The task sizes range from 1 to 10, where 1 is the easiest task and 10 is the most difficult task. Tasks with size 1-4 are categorised as easy tasks, 5-7 are intermediate tasks and 8-10 are complex tasks. In this model, it is assumed that each task can be worked on by a single developer agent or a pair of developer agents. A task has a state with four possible values: to do (task has not been worked yet); in progress (task currently being worked on); to be verified (task completed but has not been verified); and done (task is completed and verified). A task that has been completed must not be verified by the same agent(s) who worked on it.

4.2.4 The Scrum board

The Scrum board shows the progress of the sprint at each time tick by showing the status of the agents, the tasks and the user stories. The Scrum board provides a platform for all agents to share and exchange information about the current sprint. The Scrum board helps the team keep all the progress transparent and is maintained by the Scrum Master.

Task allocation process

At the beginning of the sprint, the Scrum Master selects tasks from the sprint backlog and sends these tasks to all the developer agents. It is assumed that the tasks in the sprint backlog are already prioritised such that the most important task is at the top of the list (Schwaber & Sutherland, 2017). Each developer agent inspects the tasks and gives its preferences for these tasks. A preference value is then used to determine which agents are able to undertake a task. This preference value is the gap between the task complexity and the agent's capability (agent capability – task complexity). Once the Scrum Master has received the developer agents' preferences, task allocation is then decided. A developer agent may work on the task independently (solo) or a pair of agents (pair) may work on the task. If the work scheme is solo, the developer agent who has indicated a preference for the task will be selected. If more than one agent is keen to work on the task, then the task is allocated to the agent with the lowest positive preference value. If agents are working in pairs, then the agent with the lowest positive preference value can work with any other agent (as discussed in Chapter 3).

If the work scheme is a pair, then a lead developer will be identified to work on the task (based on preference value) and an appropriate pair will be selected (as described in detail in Chapter 3). In both cases, the task is identified before deciding which agent(s) take(s) the task. In this work, there are three main types of working schemes: agent working on a task independently (solo); two agents must work on a single task (must pair); and agents that can work either solo or in pairs (intelligent/voluntary pair).

During the sprint, when a task is completed and agents become idle, the Scrum Master advertises the next task to the idle agents and the task allocation process is repeated. The Scrum Master updates the Scrum board at each time tick. The task verification must not be verified by the same developer agent and the user story must be verified by a product owner.

4.3 Simulating a Single Sprint

Before the start of a sprint, the agents and the user stories are generated. The user stories are then broken into smaller tasks, prioritised and stored in the sprint backlog. Assuming that a single agent always works on a single task, the single sprint works as follows: When a sprint starts, the Scrum Master agent selects tasks from the sprint backlog and allocates the tasks to the developer agents based on their preferences. Each agent updates its status at every time tick. When an agent has completed a task, a task is marked as 'to be verified' and the agent's status is set to idle. At this stage, the agent can request another task from the Scrum Master. The Scrum board is updated at every time tick to show the status of the agents and tasks. This process is repeated until the end of the sprint. During the sprint, when all tasks belonging to a user story are completed, the user story needs to be verified by the product owner who is external to the developer agent team. If the product owner is satisfied with the user story, it will be marked as completed. Otherwise, if the product owner is not satisfied with the user story, the tasks that belong to that user story will be put back into the sprint backlog. Each will then be given a remedial task, named as a review task (number) where the size of the remedial task will be smaller than its original task size. At the end of the sprint, any remaining tasks in the sprint backlog will be moved to the product backlog to be worked on in the next sprint. If agents are working in pair, an available agent has to wait for another agent to become available before they can work on the task. In this simulation, it is possible for tasks to be worked by individual agents and paired agents.

To simulate the software development project, multiple sprints will be run until all the user stories in the product backlog are verified and marked as completed. Figure 4.2 shows the algorithm for a single sprint and how task and agent status are updated in solo strategy. The pair strategy shares similar process but needs two developers to work and verify on a task.

```
Randomly Create n tasks;
Create m working agents;
Setup maximum Sprint time t;

While not end of the Sprint and all the tasks are not completed;

   Allocate tasks to IDLE agents;
   Update Agent Status to Working;
   Update Task Status TO IN PROGRESS ;

   For Each Allocated Task;
         If not completed;
         Continue Working on it;

         If completed;
         Update Task Status to To Be Verified;
         Update Agents Status to IDLE;

         If Verified Done;
         Update Task Status to Done;
         Update Agent Status to IDLE;
End of Sprint;
```

Figure 4.2 Example of Solo task allocation algorithm

4.4 Implementation

The simulation system is developed based on the Java Agent Development framework (JADE). This framework defines agents based on their behaviour, where each agent can be designed to have various behaviours and attributes (Bellifemine et al., 2007). JADE is a Java-based platform that can be used to develop multi-agent system. It is widely used in telecommunication industry and is not limited to modelling and simulation only.

The communication between the developer agents and the Scrum Master agent is facilitated through the message passing using the standard FIPA-ACL. The Scrum Master agent can communicate with all working agents to identify whether their working status is busy or idle, know the preference value for the task, allocate tasks to each agent, form pairs based on task requirements and allocate a task to the pair. This simulation is flexible as it allows the designer to specify the number of agents, agent's capabilities, user stories, tasks and its complexity, duration of the sprint and whether task should be worked on individually or in pair. Using this simulation, various information can be collected at each time tick such as the time taken to complete a task, who worked on the task, when the task was started,

when the task was completed. Information about agent working time and idle time can also be obtained.

4.5 Summary

This chapter described the design and implementation of a multi-agent system to simulate and evaluate strategies for the Agile software development process using Scrum. The Scrum environment is populated by the Scrum Master agent and developer agents and they worked together to deliver increments by completing a list of prioritised tasks from the sprint backlog. The task allocation process was also described where the emphasis was on making sure that the tasks were allocated to the most suitable developer agent (or pair of agents). These agents communicate and share information using the Scrum board. To complete the task, the developer agents can either work independently, in pairs or a combination of both. This simulated environment is used as a platform to execute those strategies and experiments that are needed to answer all the research questions described in Chapter 2.The next chapter describes the results of experiments that we have conducted to evaluate those strategies outlined in Chapter 3.

Chapter 5 Results and Discussion

1.9 Introduction

In this chapter the results of four experiments are presented and discussed: fixed testing, random testing, conflict modelling and multi-sprint modelling. The purpose of these experiments is to investigate the impact of varying strategies on the Scrum team's performance.The fixed experiments are categorised into 48 different environments with five or six agents with varying capabilities and varying task complexities. In each environment, each strategy is run 10 times. This is because we want to observe the average value for each metric designed in the research. Since there are 48 context and each context there are 7 strategies that need to be tested, it was decided to run each strategy in each context 10 times . Random testing provides a real-world example to show how these strategies perform in the real-world setting and to identify the best strategy in terms of average completion time, work efficiency and average effort time. Conflict modelling provides a perspective on the consistency of the selected strategy performance. Multi-sprint modelling shows how these strategies can be implemented in real world setting.

The performance of the different strategies (solo, must pair and intelligent pair) are evaluated based on effort time, idle time, completion time, person hours and work efficiency, as defined in Chapter 3 Section 3.2. In this experiment, we refer to the pair's capability as the capability of the stronger agent (referred to as the lead agent) in the pair.

5.2 Fixed numbers of agents (varying capabilities) and fixed numbers of tasks (varying complexities)

The next sections describes the results obtained for the the fixed environment. This experiment addresses RQ1, RQ2, RQ3 and RQ4.These results are presented in terms of average completion time (ACT), average effort time (AEF), average idle time (AIT) and work efficiency (WE) for each of the seven strategies.

5.2.1 Five novice agents working on tasks with evenly distributed complexities

In this experiment, the team comprised of five agents where there were more novice agents than intermediates and experts (three novices, one intermediate and one expert). This team of agents was assigned to work on evenly distributed task complexities based on Scenarios 1, 2 and 3 in Table 5.1 (Scenario 1 has 10 easy tasks, 20 intermediate tasks and 20 difficult tasks; scenario 2 has 20 easy tasks,

10 intermediate tasks and 20 difficult tasks; scenario 3 has 20 easy tasks, 20 intermediate tasks and 10 difficult tasks).

Table 5.1 Five novice agents working on tasks with evenly distributed complexities

Strategy	Scenario	ACT	AEF	AIT	WE
Solo	1	148.40	554.50	135.50	2.47
	2	144.20	493.90	183.80	2.58
	3	131.80	440.20	170.30	2.53
	Average	**141.47**	**496.20**	**163.20**	**2.53**
MP1	**COULD NOT COMPLETE**				
IP1	**COULD NOT COMPLETE**				
MP2	1	134.50	460.00	193.50	2.24
	2	132.10	440.00	201.50	2.36
	3	127.70	459.20	168.20	2.46
	Average	**131.43**	**453.07**	**187.73**	**2.35**
IP2	1	111.30	409.10	128.40	1.86
	2	108.90	384.20	141.30	1.94
	3	101.70	410.60	83.00	1.96
	Average	**107.30**	**401.30**	**117.57**	**1.92**
MP3	1	141.00	500.20	184.00	2.35
	2	132.50	455.80	186.50	2.37
	3	127.50	472.00	145.60	2.45
	Average	**133.67**	**476.00**	**172.03**	**2.39**
IP3	1	125.70	529.50	56.60	2.10
	2	119.30	477.70	73.20	2.13
	3	101.50	441.90	43.10	1.95
	Average	**115.50**	**483.03**	**57.63**	**2.06**

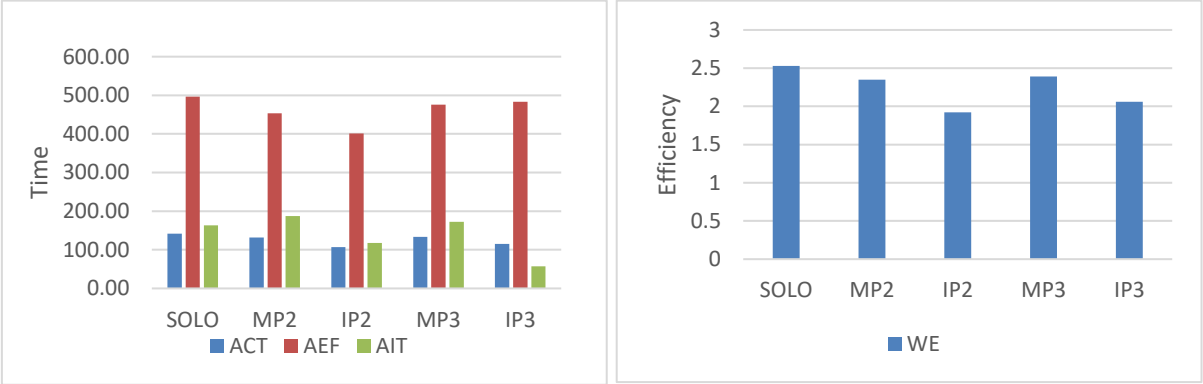
Table 5.1 shows the average completion time (ACT), average effort time (AEF), average idle time (AIT) and work efficiency (WE) for five agents for scenarios 1, 2, and 3. It can be seen that IP2 recorded the best average work efficiency for the three scenarios (1.92) followed by IP3 (2.06). IP2 also recorded the shortest completion time (107.30) and the shortest effort time (401.30), while IP3 recorded the shortest idle time (57.63). IP2 restricted the pair to work on tasks that had the same level of complexity as the pair's capability, while IP3 allowed a pair to take tasks with complexity higher than their capability (which may result in time penalties) that took longer to complete but with a shorter idle time. Overall, IP2 and IP3 performed better than the solo and the must pair strategies (MP2, MP3).

MP2 and MP3 did better than the solo because, whenever possible, it used positive pairing. Most pairings were more likely to be positive pairings, such as novice-novice, novice-intermediate and novice-expert as these pairings can work on easy, intermediate and expert tasks, respectively.

The worst work efficiency was recorded by the solo strategy, which had an average work efficiency of 2.53. This was expected because some of the agents needed to work on a task that had a higher complexity level than their capabilities, and this led to additional time, which affected completion and effort times.

MP1 and IP1 could not be completed because some of the tasks had higher levels of complexities than the pair capability (this strategy required that the pair must only work on a task complexity that was equal to or lower than the pair's capability). Figure 5.1(a) shows the performance of the strategies in terms of the average completion time, average effort time and average idle time and Figure 5.1 (b) shows the performance of the strategies in terms of work efficiency.

Figure 5.1 Comparison of strategies for five novice agents working on evenly distributed tasks



(a) (b)

5.2.2 Six novice agents working on task with evenly distributed complexities

The summary in Table 5.2 shows the results obtained when using six agents in a team (where there were three novices, two intermediates and one expert) while Figure 5.2 shows the performance of each strategy based on average completion time, average effort time and average idle time and work efficieny.. Here, MP3 recorded the best average work efficiency of the three scenarios (2.01), followed by IP3 (2.03). The worst work efficiency was recorded by the solo strategy with an average efficiency of 2.29 (see Figure 5.2 (b)). This was as expected because some agents needed to work on a task that had a higher complexity than their capability and, hence, its completion and average effort times were longer.

MP3 and IP3 have similar completion times (MP3 with 93.67 and IP3 with 95.13) and average idle (MP3 with 32.67 and IP3 with 34.03). However, MP3 recorded a higher average effort time compared to IP3. Since there were six agents in the team, MP3 allowed these agents to work in pairs by giving priority to positive pairing (either same level pairing or cross level pairing). The same level pairings that

occurred here were positive pairings, such as novice-novice working on an intermediate task and intermediate-intermediate working on a complex task. There was less opportunity for negative pairing (expert-expert) to take place due to the lower numbers of expert agents in the team. This was the same case with IP3 where some tasks were worked on by a solo agent.

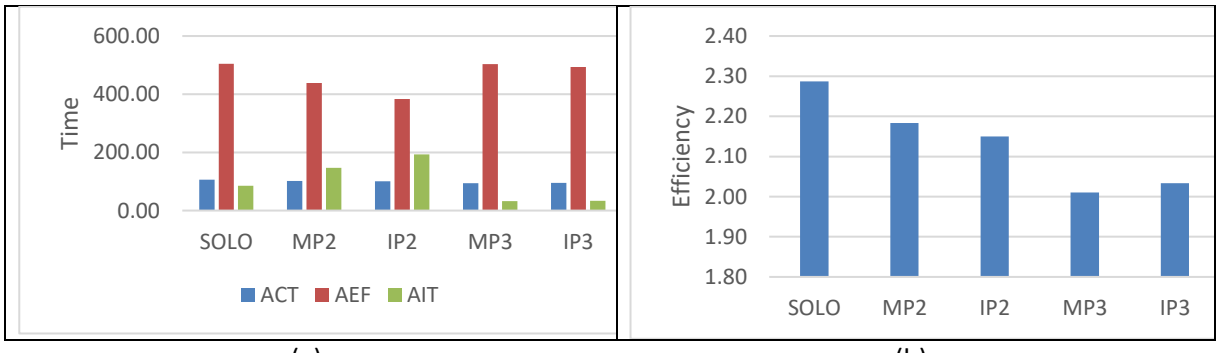
MP2 and IP2 did not perform well in this experiment because these pairs were only allowed to work on tasks that had the same complexities as the capability of the pairs. It can be seen that MP2 has an idle time of 147.20 while IP2 has an idle time of 192.57. This seemed to indicate that the available agents had to wait longer for the right pair to become available before working on the task. MP2 had a lower average idle time than IP2 because there were always two agents available to pair (as there were six agents in the team) whereas, in IP2, an available agent needed to form a positive pair, which took time.

As before, the solo strategy did not perform well in this setting where it had the highest completion time (106.53) and the highest effort time (504.37). This was because there were many agents working on tasks with complexities higher than their capabilities. This meant that they required additional time to complete the tasks which drastically reduced the idle times but increased the effort times and the completion time. As can be seen, the average idle time for solo strategy (84.93) was lower than IP2 (192.57) and MP2 (147.20).

Table 5.2 Six novice agents working on tasks with evenly distributed complexities

Strategy	Scenario	ACT	AEF	AIT	WE
Solo	1	111.80	554.00	73.00	2.24
	2	108.20	505.40	96.80	2.32
	3	99.60	453.70	85.00	2.30
	Average	**106.53**	**504.37**	**84.93**	**2.29**
MP1	**COULD NOT COMPLETE**				
IP1	**COULD NOT COMPLETE**				
MP2	1	107.90	443.20	180.20	2.16
	2	107.00	423.00	195.00	2.29
	3	91.00	448.00	66.40	2.10
	Average	**101.97**	**438.07**	**147.20**	**2.18**
IP2	1	107.80	390.10	232.70	2.16
	2	108.10	365.60	259.00	2.32
	3	85.20	393.20	86.00	1.97
	Average	**100.37**	**382.97**	**192.57**	**2.15**
MP3	1	97.40	526.60	34.40	1.95
	2	94.80	501.80	38.00	2.03
	3	88.80	481.20	25.60	2.05
	Average	**93.67**	**503.20**	**32.67**	**2.01**
IP3	1	105.00	542.30	34.40	2.10
	2	96.70	492.10	41.10	2.07
	3	83.70	445.70	26.60	1.93
	Average	**95.13**	**493.37**	**34.03**	**2.03**

Figure 5.2 Comparison of strategies for six novice agents working on evenly distributed tasks



(a) (b)

5.2.3 Five and six novice agents working on tasks with evenly distributed complexities

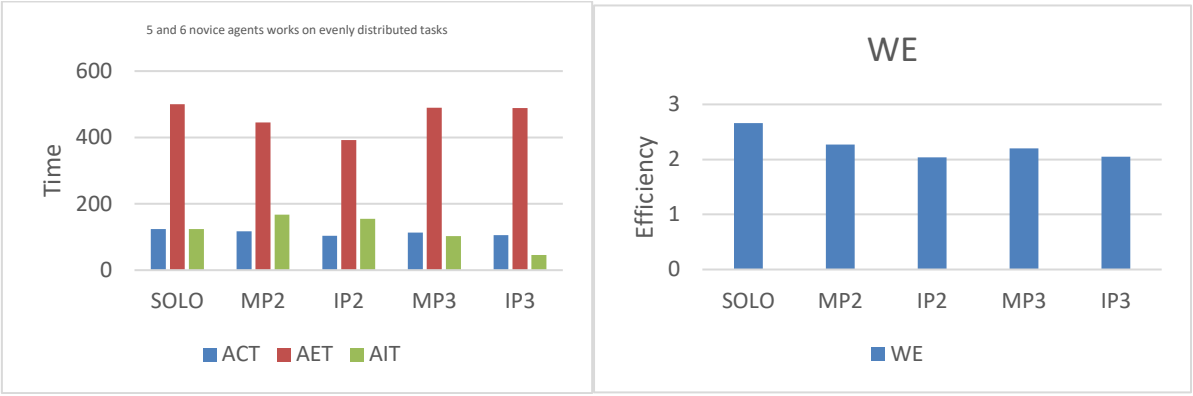
The summarised results for five and six agents are shown in Table 5.3 and Figure 5.3 shows the corresponding bar charts. When taken together it can be seen that IP2 and IP3 performed well in terms of average work efficiency, of 2.04 and 2.05, respectively. Both strategies have the two shortest completion times (103.83 and 105.32). This was because IP strategies only promoted positive pairing and no negative pairing was possible. As shown, MP2 and MP3 did not perform well in this environment. However, a closer inspection showed that MP2 and MP3 achieved their best work efficiency when there were six agents (even number) in the team as these MP strategies always had two agents working together.

The solo strategy performed the worst when compared to the must pair and intelligent strategies, as shown in Figure 5.3, which seemed to suggest that pairing was more desirable. For example, novice-novice pairs may work on an intermediate task (which has a complexity higher than novice), which was not possible with the solo strategy.

Table 5.3 Five and six novice agents working on tasks with evenly distributed complexities

Strategy	# of Agents	ACT	AET	AIT	WE
SOLO	5	**141.47**	**496.20**	**163.20**	**2.53**
	6	106.53	504.37	84.93	2.29
	Average	**124.00**	**500.28**	**124.07**	**2.41**
MP2	5	131.43	453.07	187.73	2.35
	6	101.97	438.07	147.20	2.18
	Average	**116.70**	**445.57**	**167.47**	**2.27**
IP2	5	107.30	401.30	117.57	1.92
	6	100.37	382.97	192.57	2.15
	Average	**103.83**	**392.13**	**155.07**	**2.04**
MP3	5	133.67	476.00	172.03	2.39
	6	93.67	503.20	32.67	2.01
	Average	**113.67**	**489.60**	**102.35**	**2.20**
IP3	5	115.50	483.03	57.63	2.06
	6	95.13	493.37	34.03	2.03
	Average	**105.32**	**488.20**	**45.83**	**2.05**

Figure 5.3 Comparison of strategies for five and six novice agents working on tasks with evenly distributed complexities



(a) (b)

5.2.4 Five and six novice agents working on easy tasks

In this environment, the majority of team members were novices and worked on many easy tasks. It can be seen from Table 5.4 that IP1 obtained the best work efficiency with 1.62, followed by IP2 with 1.65 and IP3 with 1.77. IP1 also had the longest completion time and average effort time. However, it had a higher average idle time because IP1 only allowed pair to work on tasks that are equal to or less than the pair's capability. IP1 did well because all the pairs were able to work on easy tasks. In this instance, there were also opportunities for novice-intermediate pairs to work on intermediate tasks or novice-expert and intermediate-expert to work on complex tasks, which was not possible for the solo strategy. This was the same for IP2 and IP3 but they did not perform as well as IP1. This is because IP2 and IP3 allowed the same level or lower level agents to work on the task, where they received more penalties than IP1.

The must pair strategies' performance for MP1, MP2 and MP3 were worse off than IP1, IP2 and IP3 because this type of pairing allowed negative pairings, which affected the work efficiency, average completion time, average effort time and average idle time for the team. It was also observed that the must pairs' performance with even numbers of agents was better than the must pair performance with odd numbers of agents, as observed in the previous experiment.

The solo strategy's average work efficiency was better than MP2 and MP3 but worse than MP1. The solo completion time was better than all the MP strategies but higher than all the IP strategies. In this environment, it can be observed that the performance of the must pairs and solo are similar in work efficiency. However, must pair strategies performed better than solo when there are even numbers of agents in the team. As can be seen in Table 5.4, when the number of agents in the team is 6, the MP3

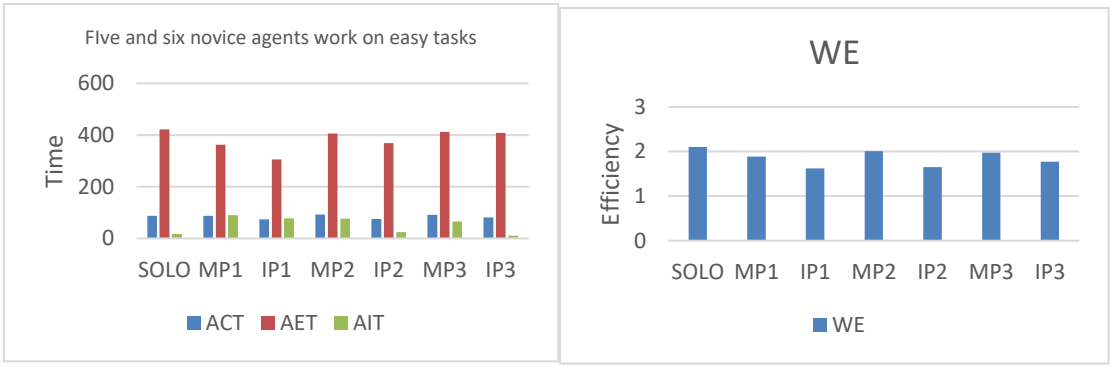
recorded an idle time of 14.00 and work efficiency of 1.76 whereas the solo strategy recorded an idle time of 13.90 and work efficiency of 1.83 which indicated that pairing still performed better than solo.

Figure 5.4 (a) shows the performance of the strategies in terms of the average completion time, average effort time and average idle time and Figure 5.4 (b) shows the performance of the strategies in terms of work efficiency.

Table 5.4 Five and six novice agents working on easy tasks

Strategy	# of Agents	ACT	AET	AIT	WE
SOLO	5	98.50	431.70	19.90	1.97
	6	76.10	413.00	13.90	1.83
	Average	**87.30**	**422.35**	**16.90**	**1.90**
MP1	5	103.80	360.00	141.20	2.08
	6	71.00	365.80	37.60	1.70
	Average	**87.40**	**362.90**	**89.40**	**1.89**
IP1	5	83.00	304.30	93.20	1.66
	6	65.30	306.10	63.20	1.57
	Average	**74.15**	**305.20**	**78.20**	**1.62**
MP2	5	109.50	411.80	117.60	2.19
	6	76.40	400.40	36.40	1.83
	Average	**92.95**	**406.10**	**77.00**	**2.01**
IP2	5	82.60	382.90	14.60	1.65
	6	68.80	354.70	35.30	1.65
	Average	**75.70**	**368.80**	**24.95**	**1.65**
MP3	5	108.80	414.20	115.80	2.18
	6	74.40	410.00	14.00	1.76
	Average	**91.60**	**412.10**	**64.90**	**1.97**
IP3	5	89.20	415.80	9.30	1.78
	6	73.10	402.20	11.10	1.75
	Average	**81.15**	**409.00**	**10.20**	**1.77**

Figure 5.4 Comparison of strategies for five and six novice agents working on easy tasks



(a) (b)

5.2.5 Five and six novice agents working on intermediate tasks

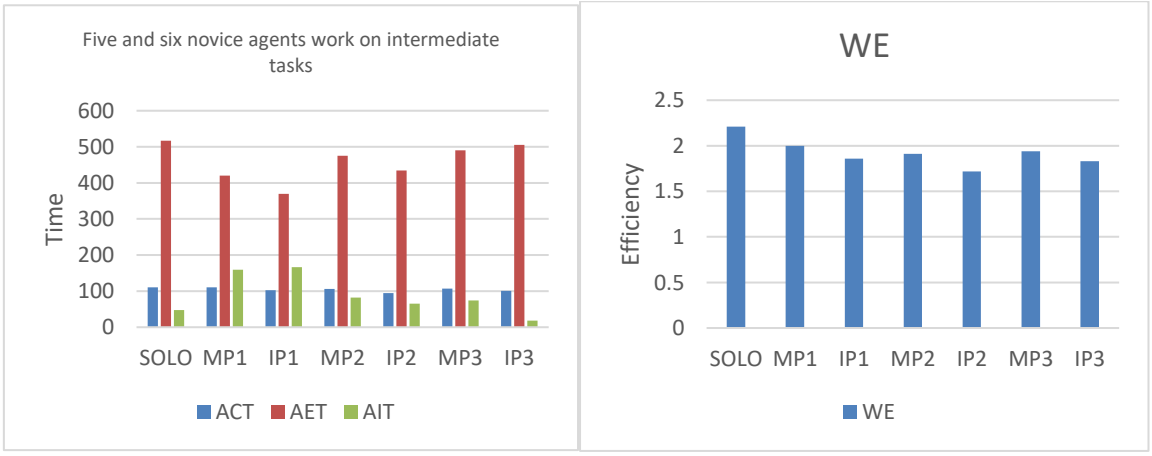
The result for many novice agents working on many intermediate tasks is shown in Table 5.5. Figure 5.5 (a) shows the performance of the strategies in terms of the average completion time, average effort time and average idle time and Figure 5.5 (b) shows the performance of the strategies in terms of work efficiency. In this context, IP2 had the best average work efficiency (1.72), followed by IP3 (1.83) and IP1 (1.86). It can be seen that the average work efficiencies of the must pair strategies (2.00 for MP1, 1.91 for MP2 and 1.94 for MP3) were worse off than the IP strategies. However, it can be seen that all the must pair strategies did well when there were six agents in the team (1.84 for MP1, 1.72 for MP2 and 1.78 for MP3). This strongly indicates that the must pair strategies only worked well for even numbers of agents. The solo strategy had the worst average work efficiency (2.00). In this environment, there were many instances where novice agents worked on intermediate tasks that took longer due to the imposition of time penalties. This was supported by the highest average effort time recorded by the solo strategy (516.50) with an average completion time of 110.25.

We observed in this environment that positive pairing had more chances of happening than negative pairing, so when there were an even number of agents, the efficiency values of MP2 (1.72) and MP3 (1.78) were much better than for the solo (1.95), so we still recommend using pair programing under such contexts. IP2 performed the best, both in the five and six agent contexts because it allowed agents work on the tasks that are in the same category as the agent capability (easy – novice, medium – intermediate and complex – expert).

Table 5.5 Five and six novice agents working on intermediate tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	122.80	521.10	56.80	2.05
	6	97.70	511.90	38.70	1.95
	Average	**110.25**	**516.50**	**47.75**	**2.00**
MP1	5	129.10	420.00	209.50	2.15
	6	92.20	420.60	108.60	1.84
	Average	**110.65**	**420.30**	**159.05**	**2.00**
IP1	5	115.00	370.00	188.70	1.92
	6	89.40	368.90	143.50	1.79
	Average	**102.20**	**369.45**	**166.10**	**1.86**
MP2	5	125.40	477.20	138.60	2.09
	6	86.00	472.80	25.40	1.72
	Average	**105.70**	**475.00**	**82.00**	**1.91**
IP2	5	107.60	440.10	85.20	1.79
	6	81.90	428.90	44.50	1.64
	Average	**94.75**	**434.50**	**64.85**	**1.72**
MP3	5	125.20	481.80	131.00	2.09
	6	88.80	498.60	16.60	1.78
	Average	**107.00**	**490.20**	**73.80**	**1.94**
IP3	5	110.40	509.90	20.40	1.84
	6	90.40	501.60	15.90	1.81
	Average	**100.40**	**505.75**	**18.15**	**1.83**

Figure 5.5 Comparison of strategies for five and six novice agents working on intermediate tasks



(a) (b)

5.2.6 Five and six novice agents working on complex tasks

. In this environment, MP3 performed best in terms of average work efficiency with a value of 2.16, which was better than IP3 (2.21), IP2 (2.36), MP2 (2.43) and solo (2.74) as shown in Table 5.6. CNC indicates that the sprint could not be completed. Figure 5.6 (a) shows the performance of the strategies in terms of the average completion time, average effort time and average idle time and Figure 5.6 (b) shows the performance of the strategies in terms of work efficiency. It achieved the shortest completion time of 135.85 and the second longest idle time (167.00). Its average work efficiency for six agents was the best among the runs with the lowest average idle time and average completion time.

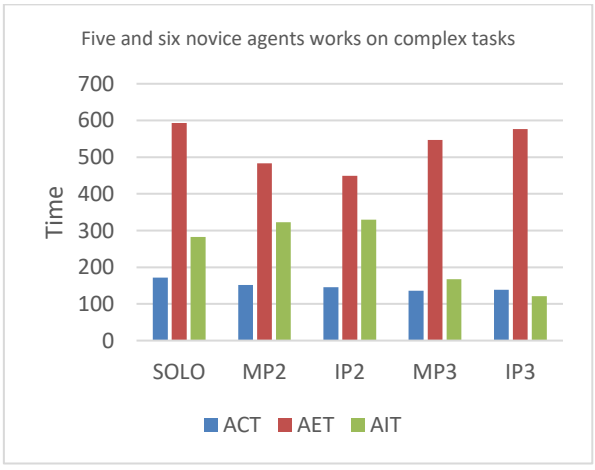
IP3 had the lowest idle time (121.25) followed by MP3 (167.00). IP2 and MP2 had the highest idle times of 329.55 and 322.85, respectively. This was expected as these two strategies only allowed paired agents to work on tasks that had the same complexities as the capabilities of the pairs and, as a consequence, these pairs of agents may have to wait until suitable tasks become available for them.

The solo strategy performed the worst, having the longest completion time (171.80), the worst average work efficiency (2.74) and the longest average effort time (593.00). This was as expected since these novice agents were forced to work on intermediate and complex tasks.

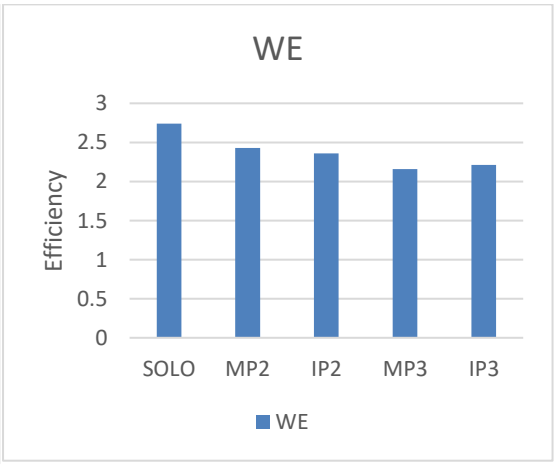
Table 5.6 Five and six novice agents working on complex tasks

Strategy	# of Agents	ACT	AET	AIT	WE
SOLO	5	199.20	581.80	355.90	2.93
	6	144.40	604.20	208.50	2.55
	Average	**171.80**	**593.00**	**282.20**	**2.74**
MP1	5	CNC	CNC	CNC	CNC
	6	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
IP1	5	CNC	CNC	CNC	CNC
	6	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
MP2	5	159.30	482.80	294.70	2.34
	6	143.10	483.60	351.00	2.52
	Average	**151.20**	**483.20**	**322.85**	**2.43**
IP2	5	148.40	449.50	273.50	2.18
	6	143.20	449.60	385.60	2.53
	Average	**145.80**	**449.55**	**329.55**	**2.36**
MP3	5	160.00	501.80	278.60	2.35
	6	111.70	592.80	55.40	1.97
	Average	**135.85**	**547.30**	**167.00**	**2.16**
IP3	5	158.00	571.50	166.00	2.32
	6	118.40	581.20	76.50	2.09
	Average	**138.20**	**576.35**	**121.25**	**2.21**

Figure 5.6 Comparison of strategies for five and six novice agents working on complex tasks



(a)



(b)

The reason why MP3 did well overall in this environment, particularly when there are six agents is due to the possible pairings that took place during the sprint. In this situation, MP3 gave preference to positive pairing in the form of intermediate-intermediate pairing and intermediate-expert pairing and very few negative pairings were observed.

A snapshot of the first run using MP3 strategy was as follows where there were no negative pairing observed:

1. Expert – intermediate working on complex task (positive pairing)
2. Expert – intermediate working on complex task (positive pairing)
3. Intermediate – intermediate working on complex task (positive pairing)
4. Intermediate – intermediate working on complex task (positive pairing)
5. Intermediate – intermediate working on complex task (positive pairing)
6. Intermediate – intermediate working on complex task (positive pairing)
7. Intermediate – intermediate working on complex task (positive pairing)
8. Intermediate – intermediate working on complex task (positive pairing)
9. Intermediate – intermediate working on complex task (positive pairing)
10. Intermediate – intermediate working on complex task (positive pairing)

Even though IP3 did not allow negative pairings, IP3 may have used solo programming when no pair can be found to work on a given task. This has resulted in its lower performance than MP3. IP1 and MP1 did not complete as there were some tasks that cannot be worked on by the team.

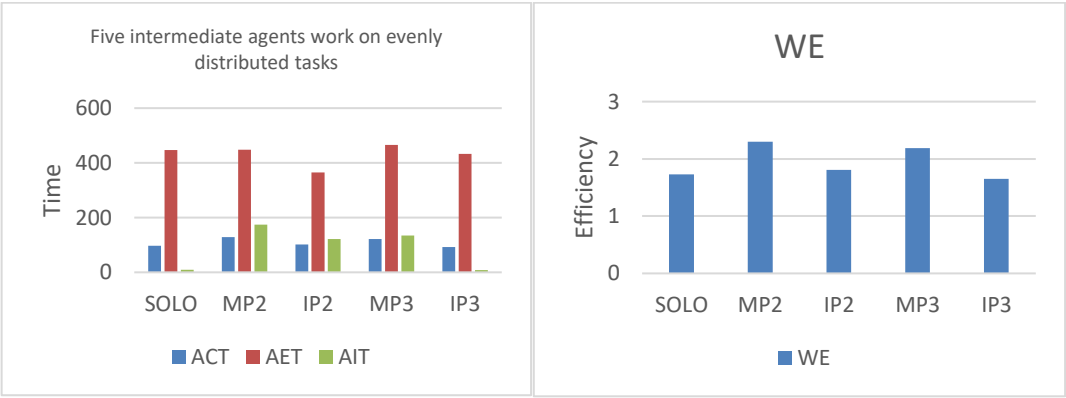
5.2.7 Five intermediate agents working on tasks with evenly distributed complexities

In this environment, IP3 performed the best in terms of average work efficiency, followed by solo (1.73), IP2 (1.81), MP3 (2.19) and MP2 (2.30) as shown in Table 5.7 and Figure 5.7. As before, MP1 and IP1 could not be completed because some of the tasks had higher levels of complexities than the pairs' capabilities The solo strategy did considerably better for all measures (worse off than IP3 but better than IP2) because most of the agents were able to do the tasks without a time penalty. Even though IP2 had the shortest average effort time, its average idle time was high as the available agents had to wait for a suitable agent to pair. IP3 performed consistently well in the three environments as it was able to balance the between working solo and pairing efficiently. On the other hand, MP2 and MP3 did not do well because they only work well when there are even number of agent in the team.

Table 5.7 Five intermediate agents working on tasks with evenly distributed complexities

Strategy	Scenario	ACT	AEF	AIT	WE
Solo	1	103.80	481.10	10.00	1.73
	2	99.50	457.10	10.20	1.78
	3	87.50	402.90	8.40	1.68
	Average	**96.93**	**447.03**	**9.53**	**1.73**
MP1	**COULD NOT COMPLETE**				
IP1	**COULD NOT COMPLETE**				
MP2	1	132.90	459.00	186.50	2.22
	2	131.40	441.60	196.40	2.35
	3	121.40	444.00	139.50	2.33
	Average	**128.57**	**448.20**	**174.13**	**2.30**
IP2	1	108.30	380.70	141.80	1.81
	2	106.40	360.60	152.40	1.90
	3	89.90	355.30	69.20	1.73
	Average	**101.53**	**365.53**	**121.13**	**1.81**
MP3	1	126.90	479.00	145.00	2.12
	2	123.20	465.20	137.40	2.20
	3	116.40	451.60	120.20	2.24
	Average	**122.17**	**465.27**	**134.20**	**2.19**
IP3	1	99.20	465.40	9.30	1.65
	2	95.90	444.40	9.30	1.71
	3	82.50	390.00	7.30	1.59
	Average	**92.53**	**433.27**	**8.63**	**1.65**

Figure 5.7 Comparison of strategies for five intermediate agents working on tasks with evenly distributed complexities



(a) (b)

5.2.8 Six intermediate agents working on tasks with evenly distributed complexities

In this context, as shown in Table 5.8 and Figure 5.8, the IP3 strategy recorded the best work efficiency (1.69), followed by the solo strategy (1.74). The MP2 work efficiency was the worst (2.19) followed by IP2 (2.04). As before, IP1 and MP1 did not complete as there were some tasks that cannot be worked on by the team.

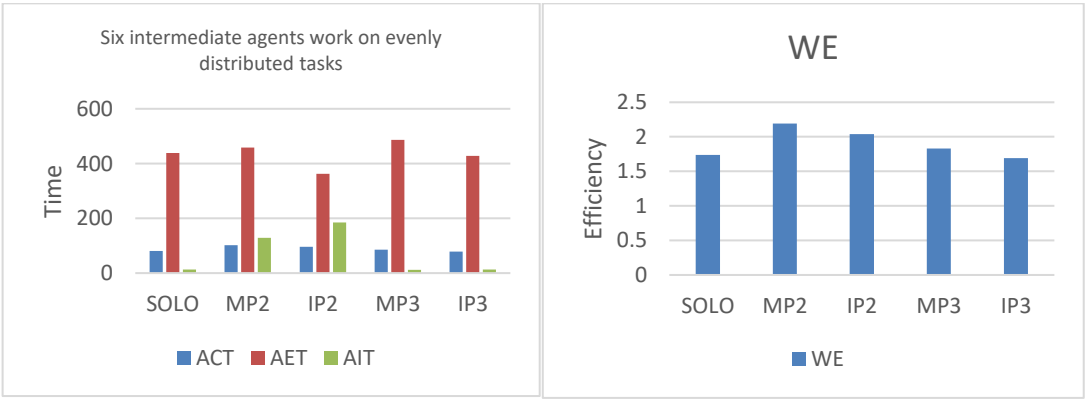
IP3 also had the shortest completion time (78.77) followed by the MP3 (85.27) and the solo strategy (81.23). All three strategies had short idle times (IP3 – 12.73, MP3 – 11.80 and solo – 13.40). However, IP2 had the shortest effort time (363.07). This is consistent with the results obtained when running the same environment with five agents in a team.

The MP3 performed well as there were six agents working in the team, most agents were intermediate agents and the task sets were evenly distributed into easy, intermediate and complex tasks. The performance of the solo strategy is also consistent with the results obtained in the same environment using five agents in a team.

Table 5.8 Six intermediate agents working on tasks with evenly distributed complexities

Strategy	Scenario	ACT	AEF	AIT	WE
Solo	1	**86.10**	**468.60**	**14.30**	**1.72**
	2	82.30	451.30	12.90	1.76
	3	75.30	397.00	13.00	1.74
	Average	**81.23**	**438.97**	**13.40**	**1.74**
MP1	**COULD NOT COMPLETE**				
IP1	**COULD NOT COMPLETE**				
MP2	1	108.60	472.80	154.80	2.17
	2	107.90	454.40	169.00	2.31
	3	90.60	447.60	64.00	2.09
	Average	**102.37**	**458.27**	**129.27**	**2.19**
IP2	1	104.20	380.30	220.90	2.08
	2	104.70	360.00	244.20	2.24
	3	78.30	348.90	88.90	1.81
	Average	**95.73**	**363.07**	**184.67**	**2.04**
MP3	1	88.60	508.80	10.40	1.77
	2	86.10	487.20	15.40	1.85
	3	81.10	463.80	9.60	1.87
	Average	**85.27**	**486.60**	**11.80**	**1.83**
IP3	1	84.10	459.50	13.30	1.68
	2	79.70	433.30	12.80	1.71
	3	72.50	390.10	12.10	1.67
	Average	**78.77**	**427.63**	**12.73**	**1.69**

Figure 5.8 Comparison of strategies for six intermediate agents working on tasks with evenly distributed complexities



(a) (b)

5.2.9 Five and six intermediate agents working on tasks with evenly distributed complexities

The summary of running the sprints using five and six agents in a team is shown in Table 5.9 and Figure 5.9. As observed, IP3 performed best for average work efficiency (1.67), followed by solo (1.74), MP2 (2.25), IP2 (1.93) and MP3 (2.01).

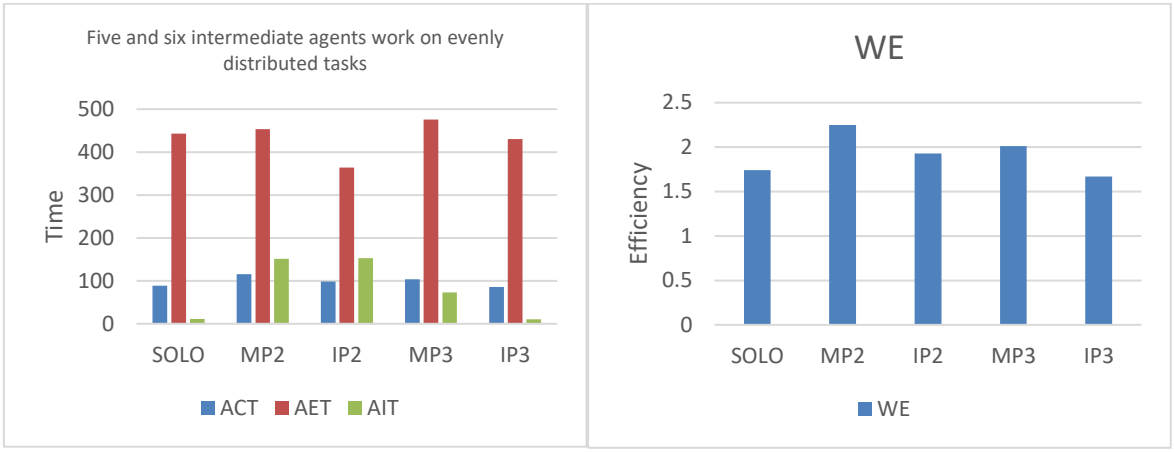
IP3 had the shortest average completion time (85.65), followed by the solo strategy (89.08) and IP2 (98.63). MP2 and MP3 had the longest completion times of 115.47 and 103.72, respectively.

Based on these observations, it can be concluded that IP3, which promoted voluntary pairing (work solo when positive pairing is not possible), worked well in this environment. Similarly, the solo strategy also performed well for all cases as most of the tasks can be worked on by the agents. IP2 also performed well in that it recorded the shortest average effort time. Both MP2 and MP3 did not do well overall as these two strategies promoted must pairs when it might have been more advantageous for some agents to work solo on some of the tasks.

Table 5.9 Five and six intermediate agents working on tasks with evenly distributed complexities

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	96.93	447.03	9.53	1.73
	6	81.23	438.97	13.40	1.74
	Average	**89.08**	**443.00**	**11.47**	**1.74**
MP2	5	128.57	448.20	174.13	2.30
	6	102.37	458.27	129.27	2.19
	Average	**115.47**	**453.24**	**151.70**	**2.25**
IP2	5	101.53	365.53	121.13	1.81
	6	95.73	363.07	184.67	2.04
	Average	**98.63**	**364.30**	**152.90**	**1.93**
MP3	5	122.17	465.27	134.20	2.19
	6	85.27	486.60	11.80	1.83
	Average	**103.72**	**475.94**	**73.00**	**2.01**
IP3	5	92.53	433.27	8.63	1.65
	6	78.77	427.63	12.73	1.69
	Average	**85.65**	**430.45**	**10.68**	**1.67**

Figure 5.9 Comparison of strategies for five and six intermediate agents working on tasks with evenly distributed complexities



(a) (b)

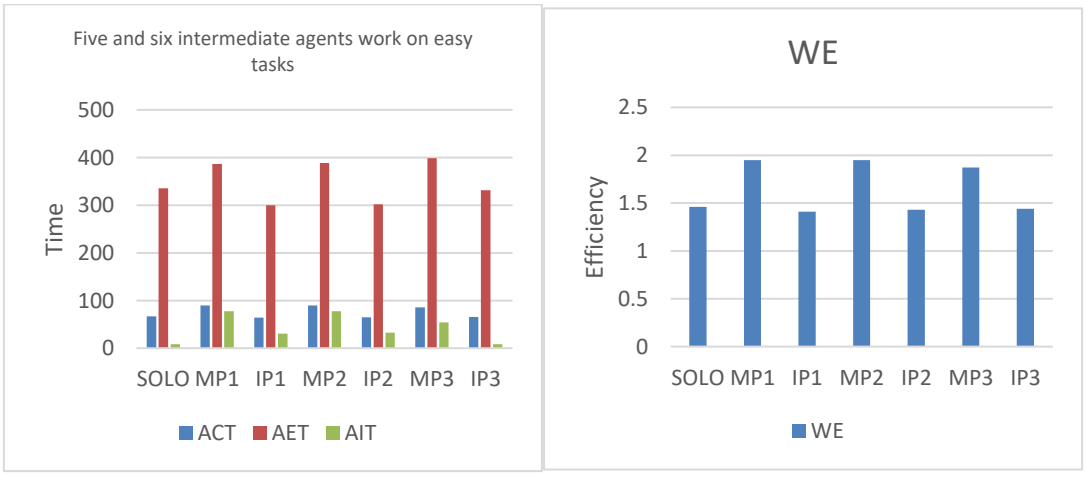
5.2.10 Five and six intermediate agents working on easy tasks

As can be seen in Table 5.10 and Figure 5.10, as expected, IP1 recorded the best average work efficiency. This was consistent with the results obtained from the many novice agents working on many easy tasks. The solo strategy also performed well as it recorded an average work efficiency of 1.46. This was as expected because the majority of agents were able to work on all the tasks. MP1, MP2 and MP3 did not perform well, as the available agents needed to wait for a suitable pair and some pairings may have resulted in negative pairing. This result also indicated that it might be better for a solo agent to work on easy tasks. This is the reason why all the IP strategies did well as there were options for the agents to work solo instead of pairing.

Table 5.10 Five and six intermediate agents working on easy tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	72.20	336.20	7.10	1.44
	6	61.40	334.10	9.60	1.47
	Average	**66.80**	**335.15**	**8.35**	**1.46**
MP1	5	102.80	380.60	114.40	2.06
	6	76.10	392.20	40.40	1.83
	Average	**89.45**	**386.40**	**77.40**	**1.95**
IP1	5	68.50	300.00	24.10	1.37
	6	60.20	300.00	37.60	1.44
	Average	**64.35**	**300.00**	**30.85**	**1.41**
MP2	5	102.90	382.40	113.10	2.06
	6	76.50	394.00	41.80	1.84
	Average	**89.70**	**388.20**	**77.45**	**1.95**
IP2	5	69.40	302.00	26.00	1.39
	6	60.70	301.60	39.00	1.46
	Average	**65.05**	**301.80**	**32.50**	**1.43**
MP3	5	98.80	386.80	97.00	1.98
	6	72.70	410.60	11.60	1.75
	Average	**85.75**	**398.70**	**54.30**	**1.87**
IP3	5	69.80	326.20	7.00	1.40
	6	61.60	335.90	9.90	1.48
	Average	**65.70**	**331.05**	**8.45**	**1.44**

Figure 5.10 Comparisons of strategies for five and six intermediate agents working on easy tasks



(a) (b)

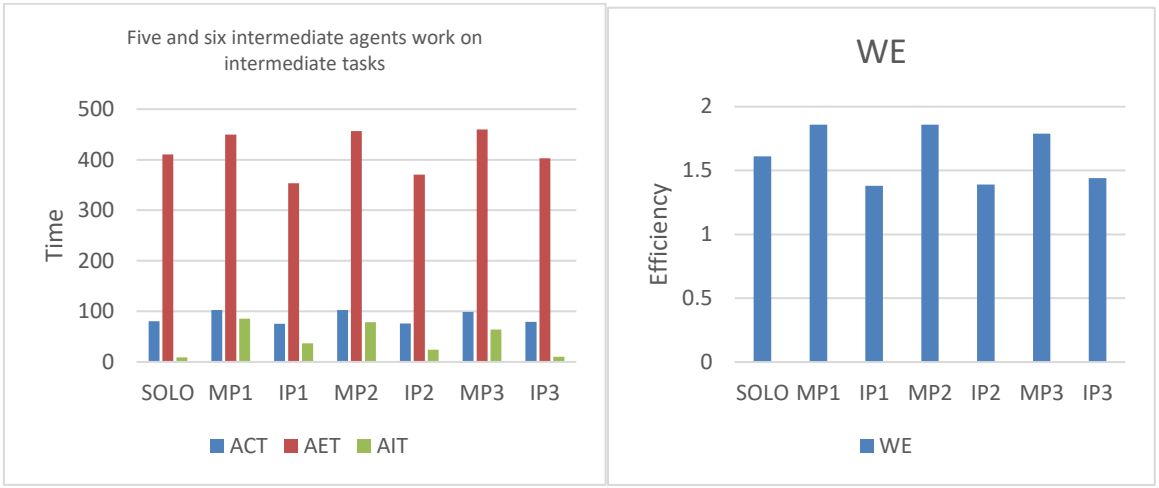
5.2.11 Five and six intermediate agents working on intermediate tasks

The results obtained in an environment where there were many intermediate agents working on many intermediate tasks is similar to the environment where there were many agents working on many easy tasks as can be seen from Table 5.11 and Figure 5.11. IP1 had the best average work efficiency (1.38) followed by IP2 (1.39) and IP3 (1.44). All the must pair strategies obtained low average work efficiencies where these values were lower than the average work efficiency for the solo strategy. This was due to this being an intermediate team working on intermediate tasks. In such contexts, pairing will result in negative pairs.

Table 5.11 Five and six intermediate agents working on intermediate tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	88.10	414.70	7.10	1.47
	6	72.90	406.20	11.20	1.46
	Average	**80.50**	**410.45**	**9.15**	**1.47**
MP1	5	119.40	445.80	133.60	1.99
	6	85.90	453.40	38.00	1.72
	Average	**102.65**	**449.60**	**85.80**	**1.86**
IP1	5	80.80	353.70	31.30	1.35
	6	70.10	353.80	42.80	1.40
	Average	**75.45**	**353.75**	**37.05**	**1.38**
MP2	5	119.90	454.40	126.40	2.00
	6	85.30	459.40	30.40	1.71
	Average	**102.60**	**456.90**	**78.40**	**1.86**
IP2	5	80.50	373.30	12.60	1.34
	6	71.40	367.30	35.90	1.43
	Average	**75.95**	**370.30**	**24.25**	**1.39**
MP3	5	114.30	446.00	114.20	1.91
	6	83.70	474.00	13.60	1.67
	Average	**99.00**	**460.00**	**63.90**	**1.79**
IP3	5	85.70	403.40	7.80	1.43
	6	72.50	402.40	12.40	1.45
	Average	**79.10**	**402.90**	**10.10**	**1.44**

Figure 5.11 Comparison of strategies for five and six intermediate agents working on intermediate tasks



(a) (b)

5.2.12 Five and six intermediate agents working on complex tasks

. As shown in Table 5.12 and Figure 5.12, IP3 performed best with an efficiency value of 1.70 in this environment and was better than MP3 (1.92), IP2 (2.32) and MP2 (2.45). This is because the intermediate team needed to work on a complex task set which is higher that the team's capability. As before, IP1 and MP1 could not complete the tasks as the agents' capabilities did not match with the task complexities. In this environment, a Type 3 strategy was able to perform well as it allowed low level agents to work on high level tasks. MP3 also worked well when there were six agents in the team.

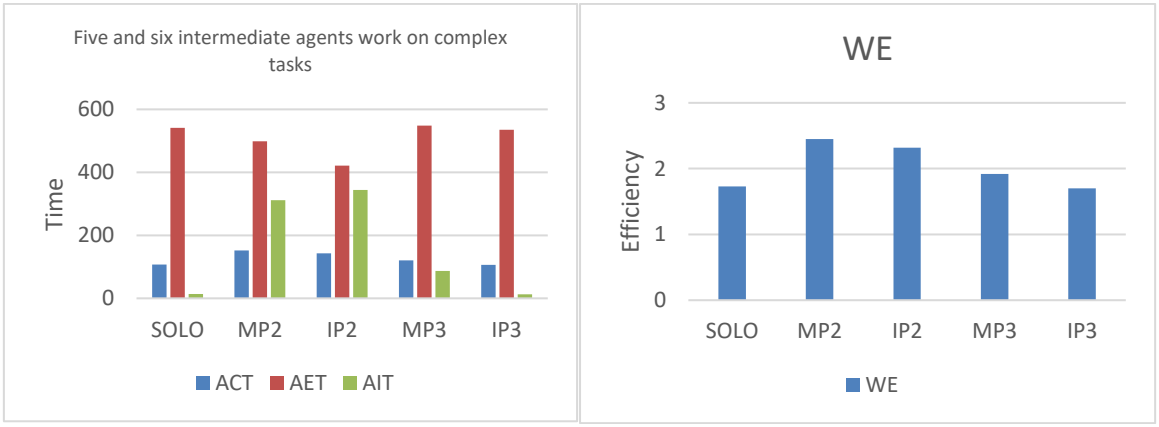
IP3 and solo performed well for completion time (106.53 and 107.50) and idle time (12.90 and 13.80). IP3 is more efficient than solo as it capitalised on pairing where two intermediate can work on a complex task which is not possible in solo. IP2 and MP2 had the highest idle times, with an IP2 idle time of 344.55 and an MP2 idle time of 311.50. This was a Type 2 strategy that only allowed agents to work on the tasks with the same category as the tasks.

This environment is very similar to many novices working on intermediate tasks, because the gaps between novice teams and intermediate tasks, and the intermediate teams and complex tasks were very similar. MP3 with six agent teams performed as expected.

Table 5.12 Five and six intermediate agents working on complex tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	117.20	545.20	12.10	1.72
	6	97.80	537.80	15.50	1.73
	Average	**107.50**	**541.50**	**13.80**	**1.73**
MP1	5	CNC	CNC	CNC	CNC
	6	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
IP1	5	CNC	CNC	CNC	CNC
	6	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
MP2	5	160.00	496.80	284.20	2.35
	6	143.70	499.40	338.80	2.54
	Average	**151.85**	**498.10**	**311.50**	**2.45**
IP2	5	143.10	421.20	275.30	2.10
	6	143.10	420.80	413.80	2.53
	Average	**143.10**	**421.00**	**344.55**	**2.32**
MP3	5	140.40	533.00	155.10	2.06
	6	100.20	564.20	19.20	1.77
	Average	**120.30**	**548.60**	**87.15**	**1.92**
IP3	5	116.10	538.90	12.20	1.70
	6	96.60	531.70	13.60	1.70
	Average	**106.35**	**535.30**	**12.90**	**1.70**

Figure 5.12 Comparison of strategies for five and six intermediate agents working on complex tasks



(a) (b)

5.2.13 Five expert agents working on tasks with evenly distributed complexities

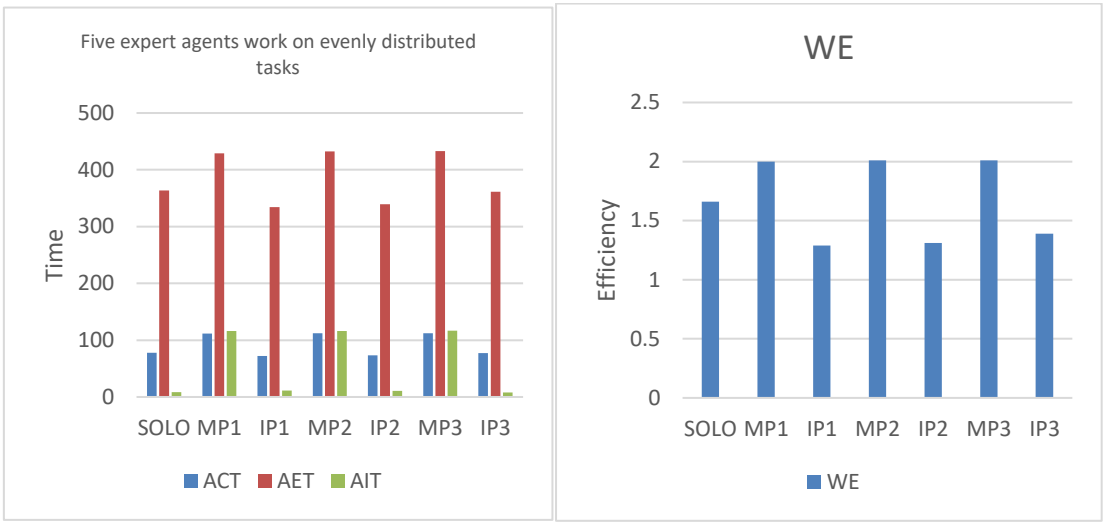
It can be seen in Table 5.13 and Figure 5.13 that IP1 recorded the best average work efficiency of 1.29, followed by IP2 (1.31), solo (1.38) and IP3 (1.39), which were better than for MP1 (2.00), MP2 (2.01) and MP3 (2.01). IP1 also recorded the shortest completion time of 72.30. The IP1 strategy had the shortest completion time (72.30), while MP2 had the longest completion time (112.47). The shortest average idle time is recorded by IP3 with 7.80.

As there were more expert agents in the team, the team was able to work on all the tasks. This is the reason why, the solo strategy did well in this environment. However, using IP1 is more advantageous as cross-pairing facilitates learning and knowledge transfer. It can also be seen that the average effort time for IP1 is lower than the average effort time for solo.

Table 5.13 Five expert agents working on tasks with evenly distributed complexities

Strategy	Scenario	ACT	AEF	AIT	WE
Solo	1	82.90	392.70	8.80	1.38
	2	78.60	363.40	7.90	1.40
	3	71.30	334.30	8.40	1.37
	Average	**77.60**	**363.47**	**8.37**	**1.38**
MP1	1	114.00	439.40	116.70	1.90
	2	112.00	421.40	123.30	2.00
	3	109.20	425.60	108.50	2.10
	Average	**111.73**	**428.80**	**116.17**	**2.00**
IP1	1	77.90	354.20	16.60	1.30
	2	71.70	332.80	9.90	1.28
	3	67.30	315.20	8.20	1.29
	Average	**72.30**	**334.07**	**11.57**	**1.29**
MP2	1	115.80	445.80	118.60	1.93
	2	112.40	426.00	121.50	2.01
	3	109.20	425.20	108.00	2.10
	Average	**112.47**	**432.33**	**116.03**	**2.01**
IP2	1	78.10	358.60	15.00	1.30
	2	73.10	338.30	10.00	1.31
	3	68.70	321.60	8.00	1.32
	Average	**73.30**	**339.50**	**11.00**	**1.31**
MP3	1	114.80	438.60	123.50	1.91
	2	109.40	419.40	116.10	1.95
	3	112.10	440.00	110.00	2.16
	Average	**112.10**	**432.67**	**116.53**	**2.01**
IP3	1	83.20	391.00	9.30	1.39
	2	75.10	355.70	6.20	1.34
	3	74.20	337.70	7.90	1.43
	Average	**77.50**	**361.47**	**7.80**	**1.39**

Figure 5.13 Comparisons of strategies for five expert agents working on tasks with evenly distributed complexities



(a) (b)

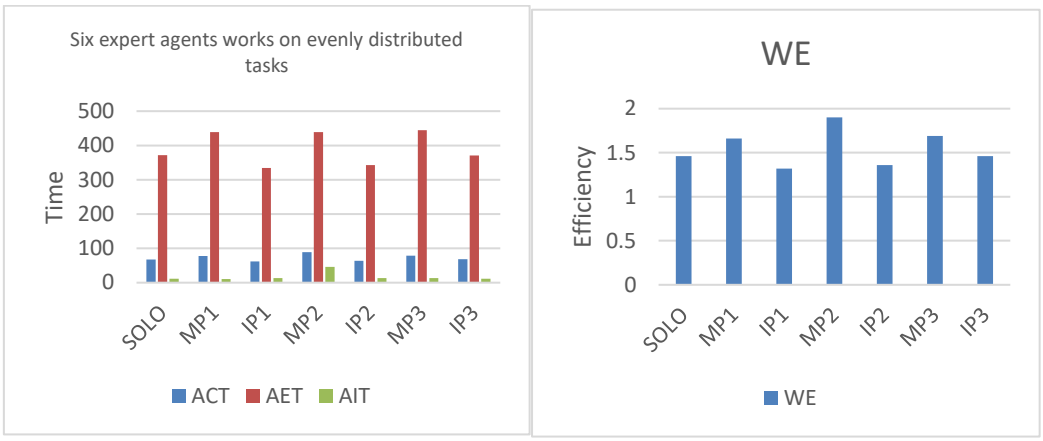
5.2.14 Six expert agents working on tasks with evenly distributed complexities

The result obtained in an environment that has six agents in a team working where majority of the agents are experts working in evenly distributed task is shown in Table 5.14 and Figure 5.14. As with the previous result with 5 agents, IP1 recorded the best work efficiency (1.32), followed by IP2 (1.36), solo (1.46) and IP3 (1.46), which were much better than MP1 (1.66), MP2 (1.67) and MP3 (1.69).

Table 5.14 Six expert agents working on tasks with evenly distributed complexities

Strategy	Scenario	ACT	AEF	AIT	WE
Solo	1	73.20	399.00	12.00	1.46
	2	67.50	376.20	10.60	1.45
	3	63.10	340.40	12.30	1.46
	Average	**67.93**	**371.87**	**11.63**	**1.46**
MP1	1	80.10	453.20	11.80	1.60
	2	76.70	436.00	10.20	1.64
	3	75.20	427.80	10.00	1.74
	Average	**77.33**	**439.00**	**10.67**	**1.66**
IP1	1	64.90	355.00	15.90	1.30
	2	61.10	333.50	12.30	1.31
	3	58.40	315.60	12.30	1.35
	Average	**61.47**	**334.70**	**13.50**	**1.32**
MP2	**1**	80.20	454.00	11.80	1.60
	2	77.10	436.40	9.80	1.65
	3	76.80	436.60	11.20	1.77
	Average	**78.03**	**442.33**	**10.93**	**1.67**
IP2	1	66.80	361.40	15.50	1.34
	2	63.40	341.60	13.10	1.36
	3	60.20	326.50	11.30	1.39
	Average	**63.47**	**343.17**	**13.30**	**1.36**
MP3	1	81.40	459.40	15.60	1.63
	2	77.30	438.20	11.40	1.66
	3	77.40	436.20	14.40	1.79
	Average	**78.70**	**444.60**	**13.80**	**1.69**
Ip3	1	71.30	393.80	10.20	1.43
	2	68.30	370.80	12.70	1.46
	3	65.10	348.90	12.40	1.50
	Average	**68.23**	**371.17**	**11.77**	**1.46**

Figure 5.14 Comparison of strategies for six expert agents working on tasks with evenly distributed complexities



(a) (b)

5.2.15 Five and six expert agents working on tasks with evenly distributed complexities

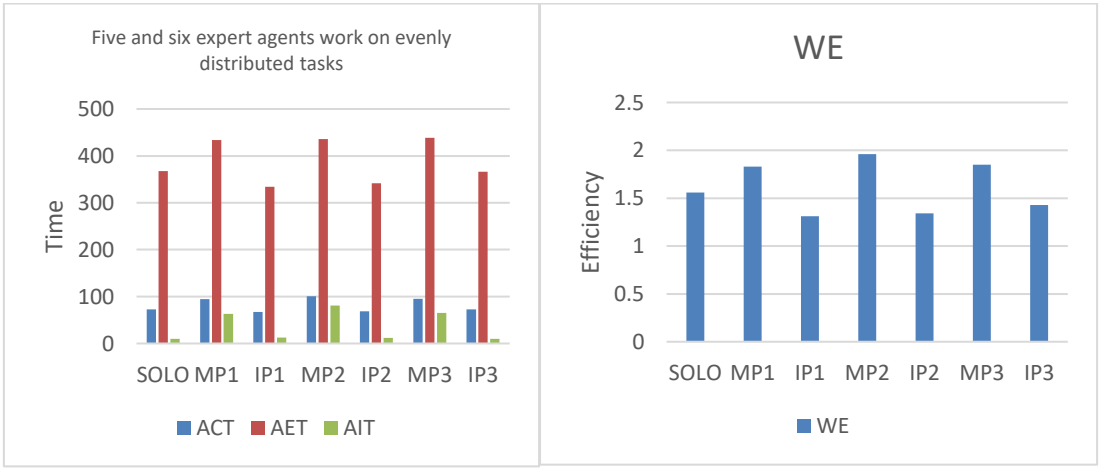
The comparison of using five and six agents in a team in an environment with evenly distributed task complexities is shown in Table 5.15 and Figure 5.15. Overall, IP1 has the best performance with average work efficiency of 1.31, followed by IP2 (1.34), solo (1.42) and IP3 (1.43), which were much better than MP1 (1.83), MP2 (1.84) and MP3 (1.85).

Based on this result, we can conclude that the must pair strategies only work well when there is an even number of agents in the team. We can observe from Table 5.13 and Table 5.14 that when there are six agents in the team the must pair strategy's efficiency is better than when there are five agents in the team. Their performance deteriorates when there are odd number of agents in the team, which means that a single agent has to wait for a pair to become available before working on a task. Moreover, the pairing may sometimes result in a negative pairing. The solo strategy performed better than IP1 and must pair strategies as there are majority of agents in the team who were able to work on the tasks. The biggest advantage can be gained from the intelligent strategies as positive pairings (same level pairing and cross pairing) are implemented and when such pairing is not possible agent may work solo on a given task. This means that average effort time is minimised and and completion time is shortened. IP3's work efficiency was lower than IP1, IP2 and solo because there may be cases where pairing might have resulted in time penalty due to its flexibility in allowing pair of agents to work on tasks where the gap between the pair capability and the task is -3.

Table 5.15 Five and six expert agents working on tasks with evenly distributed complexities

Strategy	# of Agents	ACT	AET	AIT	WE
SOLO	5	77.60	363.47	8.37	1.38
	6	67.93	371.87	11.63	1.46
	Average	**72.77**	**367.67**	**10.00**	**1.42**
MP1	5	111.73	428.80	116.17	2.00
	6	77.33	439.00	10.67	1.66
	Average	**94.53**	**433.90**	**63.42**	**1.83**
IP1	5	72.30	334.07	11.57	1.29
	6	61.47	334.70	13.50	1.32
	Average	**66.89**	**334.39**	**12.54**	**1.31**
MP2	5	112.47	432.33	116.03	2.01
	6	78.03	442.33	10.93	1.67
	Average	**95.25**	**437.33**	**63.48**	**1.84**
IP2	5	73.30	339.50	11.00	1.31
	6	63.47	343.17	13.30	1.36
	Average	**68.39**	**341.34**	**12.15**	**1.34**
MP3	5	112.10	432.67	116.53	2.01
	6	78.70	444.60	13.80	1.69
	Average	**95.40**	**438.64**	**65.17**	**1.85**
IP3	5	77.50	361.47	7.80	1.39
	6	68.23	371.17	11.77	1.46
	Average	**72.87**	**366.32**	**9.79**	**1.43**

Figure 5.15 Comparison of strategies for five and six expert agents working on tasks with evenly distributed complexities



(a) (b)

5.2.16 Five and six expert agents working on easy tasks

The result when there are five and six agents working in a team in an environment where there are a lot of easy tasks is shown in Table 5.16 and Figure 5.16. In this environment, most of the agents were able to work on all the tasks without time penalty as their capabilities were a lot higher than the task complexities. IP1 recorded the best average work efficiency of 1.28 (as expected), followed by IP2 (1.30), solo (1.35) and IP3 (1.34), which were much better than MP1 (1.85), MP2 (1.85) and MP3 (1.85).

Table 5.16 Five and six expert agents working on easy tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	65.90	309.40	7.90	1.32
	6	57.20	314.90	9.90	1.37
	Average	**61.55**	**312.15**	**8.90**	**1.35**
MP1	5	99.10	385.60	99.40	1.98
	6	71.50	401.60	12.80	1.72
	Average	**85.30**	**393.60**	**56.10**	**1.85**
IP1	5	63.70	300.00	7.00	1.27
	6	53.90	300.00	7.50	1.29
	Average	**58.80**	**300.00**	**7.25**	**1.28**
MP2	5	98.80	383.60	98.90	1.98
	6	71.20	400.00	13.80	1.71
	Average	**85.00**	**391.80**	**56.35**	**1.85**
IP2	5	63.90	300.00	6.90	1.28
	6	54.40	300.00	9.40	1.31
	Average	**59.15**	**300.00**	**8.15**	**1.30**
MP3	5	99.00	385.00	99.30	1.98
	6	71.40	402.80	11.60	1.71
	Average	**85.20**	**393.90**	**55.45**	**1.85**
IP3	5	65.30	307.10	6.30	1.31
	6	56.70	314.70	8.00	1.36
	Average	**61.00**	**310.90**	**7.15**	**1.34**

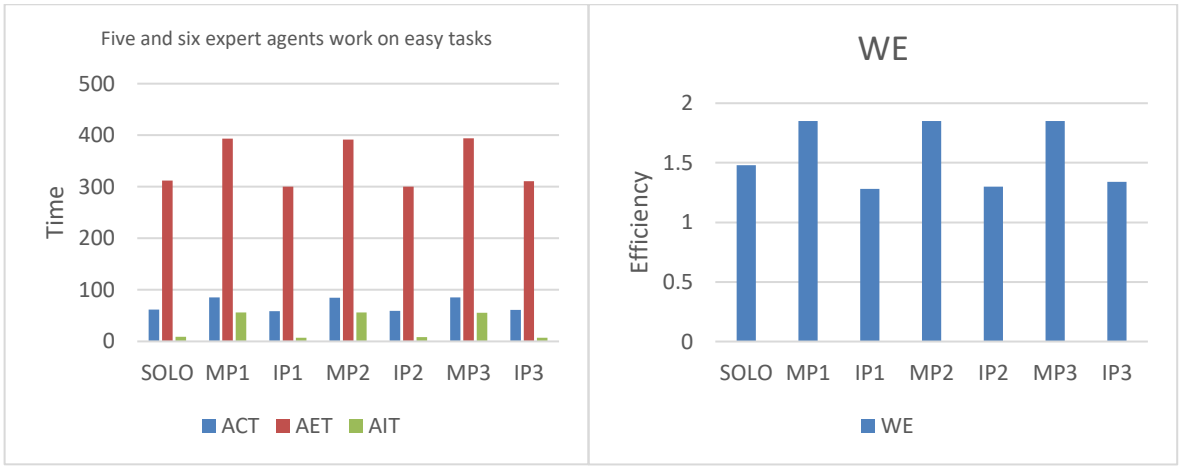
The poor performance of must pair strategies can be explained by exploring the pairings that occurred on a single run for a team of six agents. The following pairings based on MP3 were captured where 9 out of 10 were negative pairings.

1. Intermediate – intermediate working on intermediate task (negative pairing)
2. Expert – expert working on easy task (negative pairing)

3. Expert - expert working on easy task (negative pairing)
4. Expert – expert working on complex task (negative pairing)
5. Expert – intermediate working on easy task (negative pairing)
6. Expert – intermediate working on easy task (negative pairing)
7. Expert – expert working on complex task (negative pairing)
8. Expert – intermediate working on easy task (negative pairing)
9. Expert – intermediate working on easy task (negative pairing)
10. Expert – intermediate working on complex task (positive pairing)

This confirms that it is not necessary to pair all the time as negative pairings may prolong the time it takes to complete a task which may lead to longer completion time, more effort time and poor work efficiency.

Figure 5.16 Comparison of strategies for five and six expert agents working on easy tasks



(a) (b)

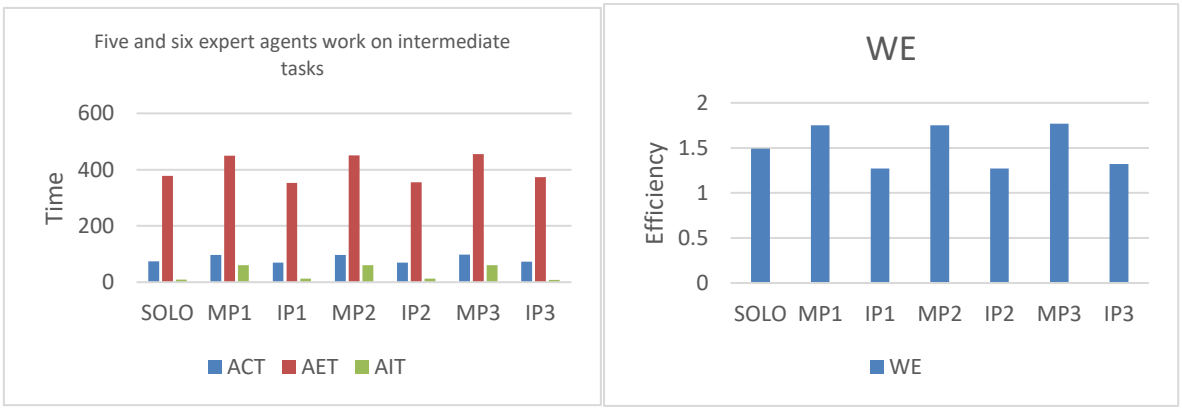
5.2.17 Five and six expert agents working on intermediate tasks

The result for this environment is shown in Table 5.17 and Figure 5.17. This environment is also similar to the previous environment where most of the agents were able to work on the tasks. As before, IP1 and IP2 has the best work efficiency (1.27), followed by solo (1.36) and IP3 (1.32), which were much better than MP1 (1.75), MP2 (1.75) and MP3 (1.77).

Table 5.17 Five and six expert agents working on intermediate tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	79.40	374.30	7.40	1.32
	6	69.40	381.40	11.00	1.39
	Average	**74.40**	**377.85**	**9.20**	**1.36**
MP1	5	112.10	440.00	109.70	1.87
	6	81.40	459.80	12.40	1.63
	Average	**96.75**	**449.90**	**61.05**	**1.75**
IP1	5	75.10	352.80	11.60	1.25
	6	64.00	352.80	13.90	1.28
	Average	**69.55**	**352.80**	**12.75**	**1.27**
MP2	5	112.10	440.00	109.30	1.87
	6	80.90	460.60	10.80	1.62
	Average	**96.50**	**450.30**	**60.05**	**1.75**
IP2	5	75.20	352.80	11.60	1.25
	6	64.60	357.40	14.00	1.29
	Average	**69.90**	**355.10**	**12.80**	**1.27**
MP3	5	112.10	440.00	110.50	1.87
	6	83.40	471.40	11.60	1.67
	Average	**97.75**	**455.70**	**61.05**	**1.77**
IP3	5	77.00	363.70	7.80	1.28
	6	68.00	382.00	9.60	1.36
	Average	**72.50**	**372.85**	**8.70**	**1.32**

Figure 5.17 Comparison of strategies for five and six expert agents working on intermediate tasks



(a) (b)

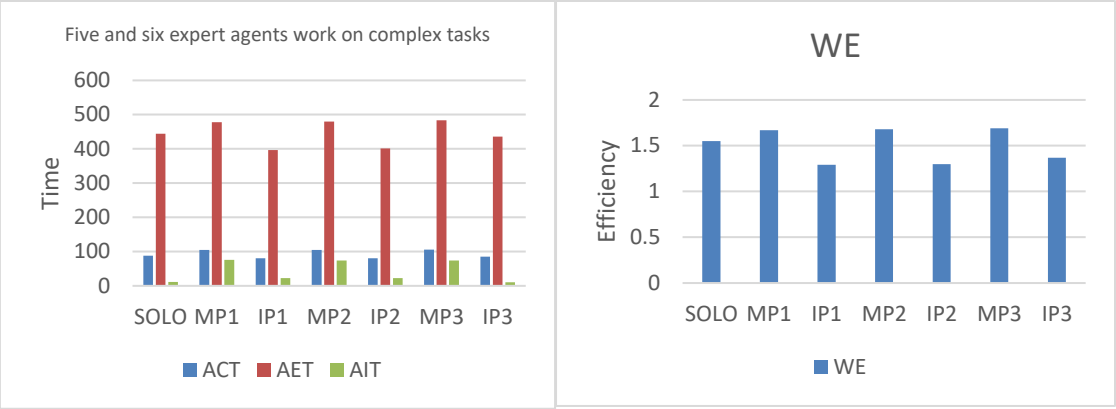
5.2.18 Five and six expert agents working on complex tasks

The result obtained when running the strategies with five and six agents (mostly experts) in an environments where majority of the tasks are complex is shown in Table 5.18 and Figure 5.18. As majority of the agents can work on all the tasks, we expect this result to be similar to the previous ones when there are many expert agents in a team. This is the reason why, IP1 recorded the best work efficiency (1.29), followed by IP2 (1.30), solo (1.42) and IP3 (1.37), which were much better than MP1 (1.67), MP2 (1.68) and MP3 (1.69).

Table 5.18 Five and six expert agents working on complex tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	92.30	432.20	9.30	1.36
	6	83.20	456.60	14.10	1.47
	Average	**87.75**	**444.40**	**11.70**	**1.42**
MP1	5	123.80	471.60	133.70	1.82
	6	86.40	484.40	18.00	1.52
	Average	**105.10**	**478.00**	**75.85**	**1.67**
IP1	5	86.10	396.30	16.60	1.27
	6	74.50	396.70	29.30	1.31
	Average	**80.30**	**396.50**	**22.95**	**1.29**
MP2	5	123.50	473.00	130.60	1.82
	6	86.90	486.80	18.00	1.53
	Average	**105.20**	**479.90**	**74.30**	**1.68**
IP2	5	85.90	399.40	14.00	1.26
	6	76.00	403.30	31.10	1.34
	Average	**80.95**	**401.35**	**22.55**	**1.30**
MP3	5	123.10	473.80	129.10	1.81
	6	88.20	492.60	18.00	1.56
	Average	**105.65**	**483.20**	**73.55**	**1.69**
IP3	5	89.40	425.20	8.80	1.31
	6	80.70	446.90	11.50	1.42
	Average	**85.05**	**436.05**	**10.15**	**1.37**

Figure 5.18 Comparison of strategies for five and six expert agents working on complex tasks



(a) (b)

5.2.19 Five agents with evenly distributed capabilities working on tasks with evenly distributed complexities

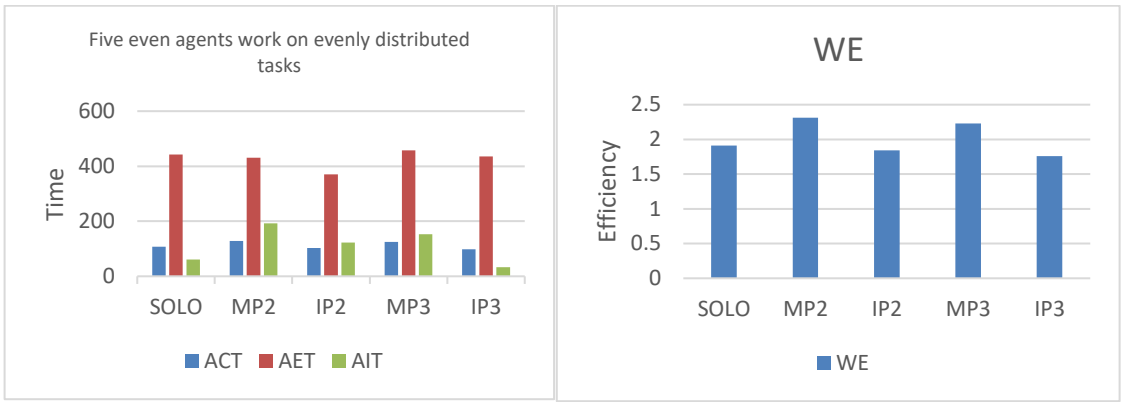
As shown in Table 5.19 and Figure 5.19, IP3 and IP2 performed best (1.76 and 1.84 respectively). IP3 also had the shortest completion time (98.40). Other strategies had longer completion times, such as solo (107.07), MP2 (128.97), IP2 (102.87) and MP3 (124.70).

IP1 and MP1 could not complete as some of the tasks had complexities higher than the agents' capabilities. IP3 did better than the other strategies as it allows for positive pairings to work on tasks that have higher complexities than the pairs. It can also be observed that IP3 has the shortest average idle time. MP2 had the longest idle time (192.70) and completion time (128.74) due to the odd number of agent in the team. Similar result can be observed for MP3 where it recorded the highest average effort time (457.93) due to the time penalty incurred in the negative pairings that took place.

Table 5.19 Five even agents working on tasks with evenly distributed complexities

Strategy	Scenario	ACT	AEF	AIT	WE
Solo	1	115.90	477.40	70.10	1.93
	2	108.80	422.20	90.90	1.94
	3	96.50	427.50	21.20	1.85
	Average	**107.07**	**442.37**	**60.73**	**1.91**
MP1	1	CNC	CNC	CNC	CNC
	2	CNC	CNC	CNC	CNC
	3	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
IP1	1	CNC	CNC	CNC	CNC
	2	CNC	CNC	CNC	CNC
	3	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
MP2	1	132.70	441.20	203.30	2.21
	2	130.50	421.20	212.30	2.33
	3	123.70	431.00	162.50	2.38
	Average	**128.97**	**431.13**	**192.70**	**2.31**
IP2	1	109.20	381.90	145.10	1.82
	2	107.80	362.40	157.60	1.93
	3	91.60	368.50	64.50	1.76
	Average	**102.87**	**370.93**	**122.40**	**1.84**
MP3	1	129.80	481.00	155.80	2.16
	2	127.00	447.40	175.50	2.27
	3	117.30	445.40	129.00	2.26
	Average	**124.70**	**457.93**	**153.43**	**2.23**
IP3	1	105.20	471.30	31.30	1.75
	2	99.60	415.00	58.80	1.78
	3	90.40	421.70	10.30	1.74
	Average	**98.40**	**436.00**	**33.47**	**1.76**

Figure 5.19 Comparison of strategies for five even agents working on tasks with evenly distributed complexities



(a) (b)

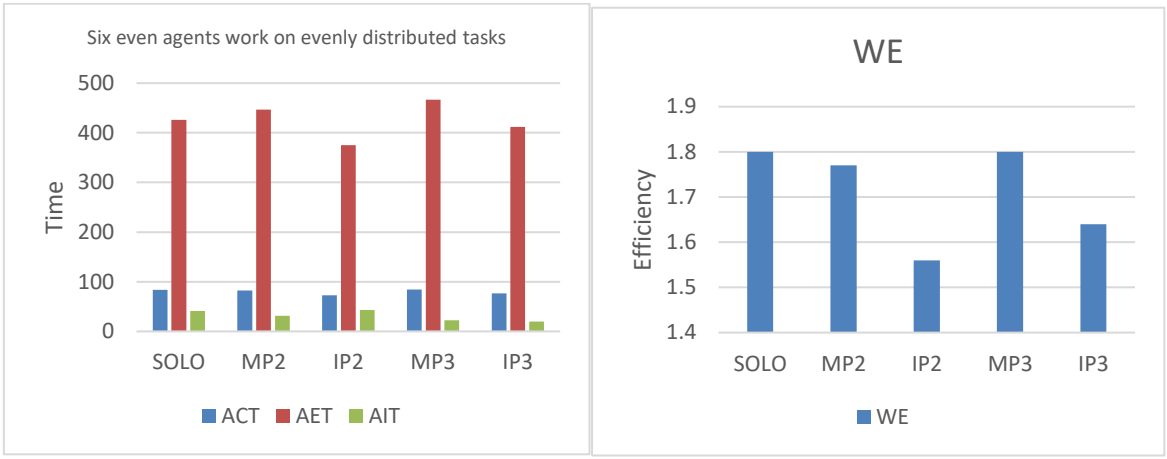
5.2.20 Six agents with evenly distributed capabilities working on tasks with evenly distributed complexities

Consistent with the previous result, it can be seen in Table 5.20 and Figure 5.20 that IP2 performed best (1.56) followed by IP3 (1.64). IP2 also had the shortest average completion time of 72.87, followed by IP3 (76.70). The other strategies had longer completion times; solo with 83.83, MP2 with 82.43 and MP3 with 84.07. As in the previous experiment, IP1 and MP1 could not complete as some of the tasks had complexities higher than the agents' capabilities.

Table 5.20 Six agents with evenly distributed capabilities working on tasks with evenly distributed complexities

Strategy	Scenario	ACT	AEF	AIT	WE
Solo	1	88.90	460.50	41.50	1.78
	2	85.70	408.70	64.10	1.84
	3	76.90	409.10	17.50	1.77
	Average	**83.83**	**426.10**	**41.03**	**1.80**
MP1	1	CNC	CNC	CNC	CNC
	2	CNC	CNC	CNC	CNC
	3	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
IP1	1	CNC	CNC	CNC	CNC
	2	CNC	CNC	CNC	CNC
	3	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
MP2	1	83.70	453.80	31.60	1.67
	2	83.10	438.80	41.80	1.78
	3	80.50	446.80	21.00	1.86
	Average	**82.43**	**446.47**	**31.47**	**1.77**
IP2	1	76.90	385.50	58.10	1.54
	2	73.70	372.00	51.00	1.58
	3	68.00	368.00	19.40	1.57
	Average	**72.87**	**375.17**	**42.83**	**1.56**
MP3	1	88.10	483.80	27.40	1.76
	2	82.00	447.60	28.20	1.75
	3	82.10	468.00	12.20	1.89
	Average	**84.07**	**466.47**	**22.60**	**1.80**
Ip3	1	83.50	446.30	19.80	1.67
	2	75.40	396.90	26.70	1.62
	3	71.20	392.80	12.70	1.64
	Average	**76.70**	**412.00**	**19.73**	**1.64**

Figure 5.20 Comparison of strategies for six agents with evenly distributed capabilities working on tasks with evenly distributed complexities



(a) (b)

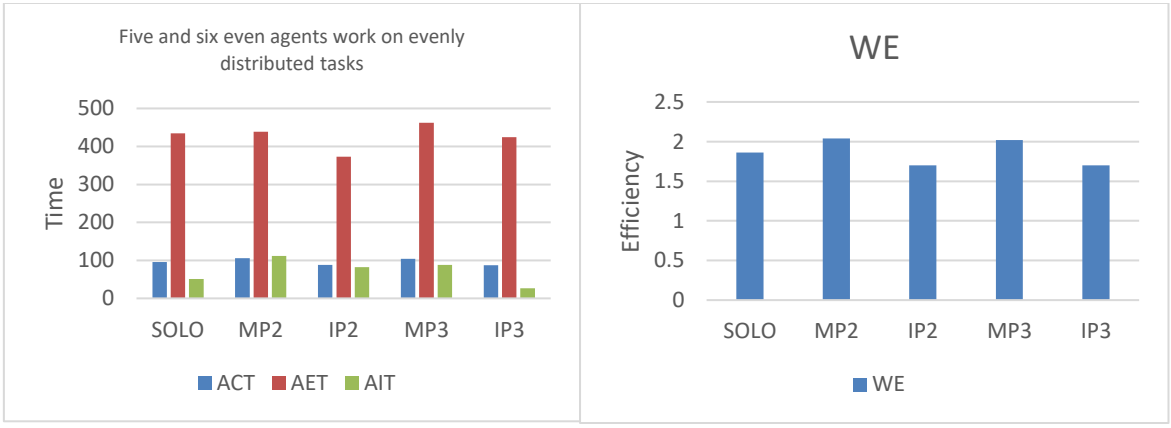
5.2.21 Five and six agents with evenly distributed capabilities working on tasks with evenly distributed complexities

The result obtained for both five and six agents in a team where their capabilities are evenly distributed and working with evenly distributed tasks is shown in Table 5.21 and Figure 5.21. Overall, IP2 and IP3 performed best with 1.70 which is better than solo (1.86), MP2 (2.04) and MP3 (2.02). IP3 also had the shortest average completion time (87.55), followed by IP2 (87.87). The other strategies had longer completion times; solo with 95.45, MP2 with 105.70 and MP3 with 104.39. As before, both must pair strategies did better when there are even number of agent in a team. It can be concluded that, in this enviroment, IP2 and IP3 are the preferred strategies as they allow more flexibilities in the tasks that can be worked by the positive pairing. In addition, no negative pairings are allowed which explained why the average completion time is much lower than the other strategies.

Table 5.21 Five and six agents with evenly distributed capabilities working on tasks with evenly distributed complexities

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	107.07	442.37	60.73	1.91
	6	83.83	426.10	41.03	1.80
	Average	**95.45**	**434.24**	**50.88**	**1.86**
MP2	5	128.97	431.13	192.70	2.31
	6	82.43	446.47	31.47	1.77
	Average	**105.70**	**438.80**	**112.09**	**2.04**
IP2	5	102.87	370.93	122.40	1.84
	6	72.87	375.17	42.83	1.56
	Average	**87.87**	**373.05**	**82.62**	**1.70**
MP3	5	124.70	457.93	153.43	2.23
	6	84.07	466.47	22.60	1.80
	Average	**104.39**	**462.20**	**88.02**	**2.02**
IP3	5	98.40	436.00	33.47	1.76
	6	76.70	412.00	19.73	1.64
	Average	**87.55**	**424.00**	**26.60**	**1.70**

Figure 5.21 Comparison of strategies for five and six agents with evenly distributed capabilities working on tasks with evenly distributed complexities



(a) (b)

5.2.22 Five and Six agents with evenly distributed capabilities working on easy tasks

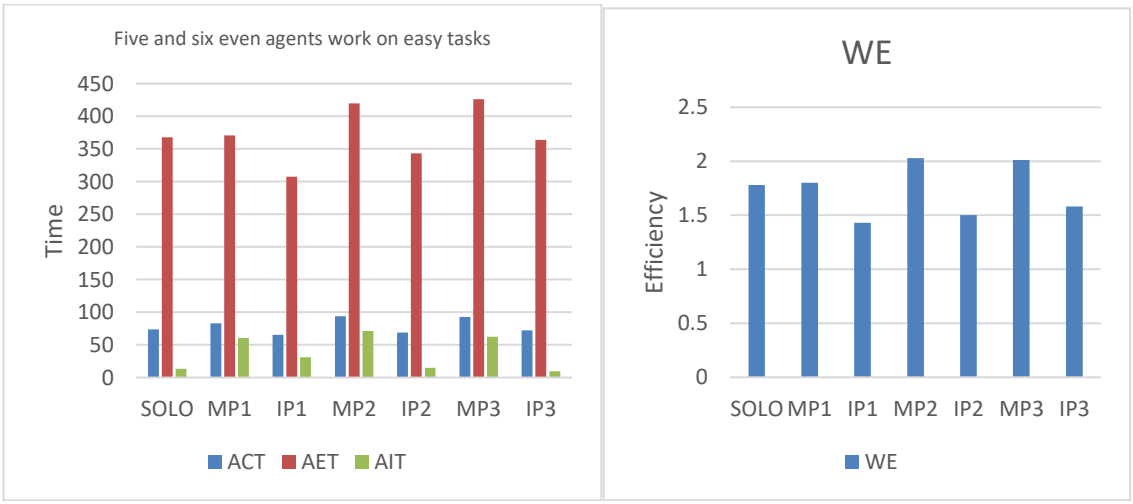
In this environment, most of the tasks can be worked by all the agents as majority of the tasks have lower complexities. It can be seen from Table 5.22 and Figure 5.22 that IP1 had the best work efficiency of 1.43 followed by IP2 (1.50), solo (1.61) and IP3(1.58), which were much better than for MP1 (1.80), MP2 (2.03) and MP3 (2.01). This is similar to the previous situations where there were more expert

agents working on easy tasks, more expert agents working on intermediate tasks and more intermediate agents working on easy tasks.

Table 5.22 Five and six agents with evenly distributed task capabilities working on easy tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	82.80	379.60	13.30	1.66
	6	64.90	355.70	13.60	1.56
	Average	**73.85**	**367.65**	**13.45**	**1.61**
MP1	5	96.80	360.00	105.00	1.94
	6	69.20	382.00	16.40	1.66
	Average	**83.00**	**371.00**	**60.70**	**1.80**
IP1	5	72.40	307.10	38.10	1.45
	6	58.30	307.60	23.70	1.40
	Average	**65.35**	**307.35**	**30.90**	**1.43**
MP2	5	114.00	421.20	130.90	2.28
	6	74.10	418.60	11.60	1.78
	Average	**94.05**	**419.90**	**71.25**	**2.03**
IP2	5	76.60	345.90	18.90	1.53
	6	61.20	340.40	10.60	1.47
	Average	**68.90**	**343.15**	**14.75**	**1.50**
MP3	5	111.40	435.80	110.60	2.23
	6	74.30	416.40	14.80	1.78
	Average	**92.85**	**426.10**	**62.70**	**2.01**
IP3	5	79.30	372.90	8.00	1.59
	6	65.10	354.60	11.40	1.56
	Average	**72.20**	**363.75**	**9.70**	**1.58**

Figure 5.22 Comparison of strategies for five and six agents with evenly distributed task capabilities working on easy tasks



(a) (b)

5.2.23 Five and six agents with evenly distributed capabilities working on intermediate tasks

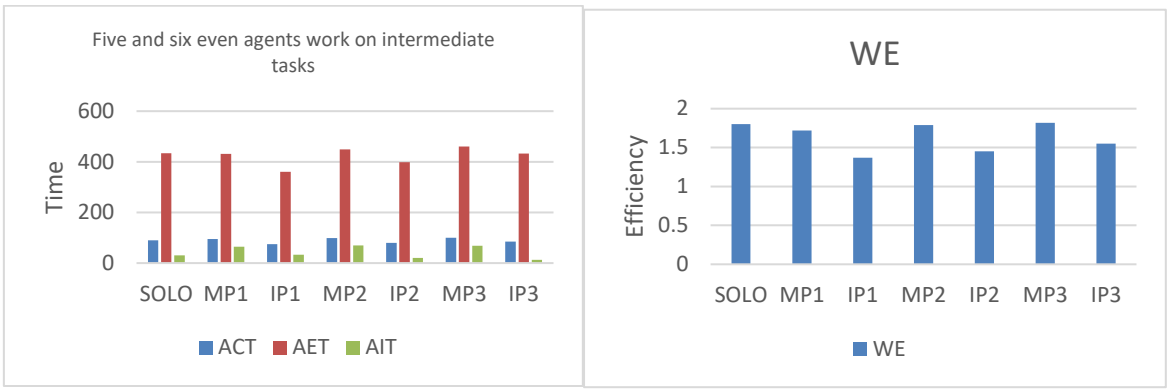
The performance of all the strategies in this environment is shown in Table 5.23 and Figure 5.23, IP1 performed the best (1.37), followed by IP2 (1.45), solo (1.64) and IP3 (1.55), which were much better than MP1 with 1.72, MP2 with 1.79 and MP3 with 1.82.

In this environment, most of the agents were able to work on all the tasks as there were equal number of novice agents, intermediate agents and expert agents (for 6 agents in a team). In cases where, a single agent was not able to work on the task, with the IP strategies, this agent can pair up with another agent. As expected, the performance of the solo strategy was acceptable as due to the even distribution in task complexities and agents' capabilities.

Table 5.23 Five and six agents with evenly distributed capabilities working on intermediate tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	99.60	438.40	35.30	1.66
	6	80.50	428.30	24.70	1.61
	Average	**90.05**	**433.35**	**30.00**	**1.64**
MP1	5	110.70	420.00	115.20	1.85
	6	78.80	442.60	14.20	1.58
	Average	**94.75**	**431.30**	**64.70**	**1.72**
IP1	5	83.90	361.20	42.20	1.40
	6	66.80	358.80	24.90	1.34
	Average	**75.35**	**360.00**	**33.55**	**1.37**
MP2	5	116.70	442.80	123.10	1.95
	6	81.40	456.00	17.00	1.63
	Average	**99.05**	**449.40**	**70.05**	**1.79**
IP2	5	87.90	400.80	24.20	1.47
	6	71.50	396.50	15.60	1.43
	Average	**79.70**	**398.65**	**19.90**	**1.45**
MP3	5	117.30	449.60	124.80	1.96
	6	83.70	472.40	13.40	1.67
	Average	**100.50**	**461.00**	**69.10**	**1.82**
IP3	5	94.90	442.30	13.00	1.58
	6	76.20	422.50	12.80	1.52
	Average	**85.55**	**432.40**	**12.90**	**1.55**

Figure 5.23 Comparison of strategies for five and six agents with evenly distributed capabilities working on intermediate tasks



(a) (b)

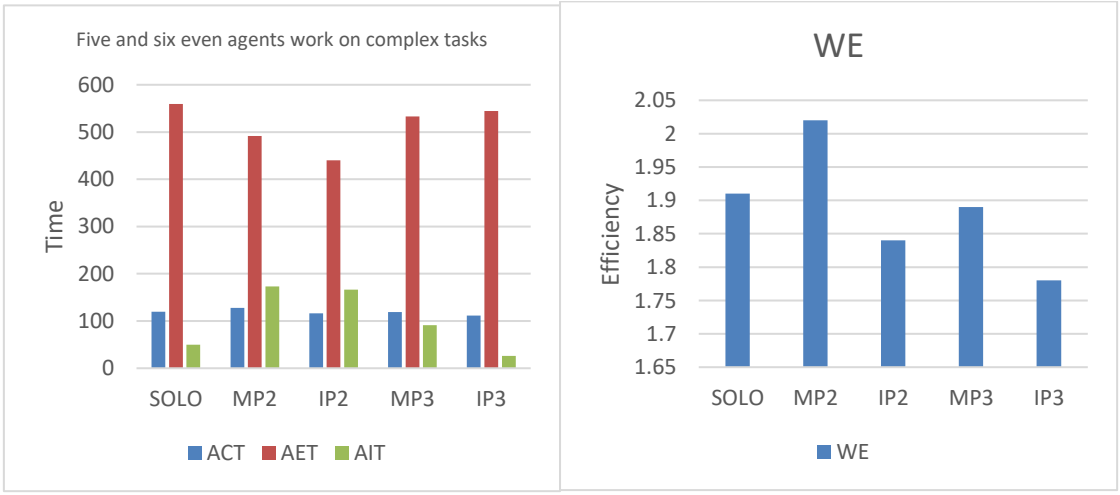
5.2.24 Five and six agents with evenly distributed capabilities working on complex tasks

In this environment, there were more complex tasks that need to be completed and this has contributed to the inability of IP1 and MP1 to complete all the tasks. As shown in Table 5.24 and Figure 5.24, IP3 performed best with average work efficiency of 1.78 followed by IP2 with 1.84. In this run, MP3 recorded a higher average work efficiency (1.89) and than solo (1.91) and shorter average completion time (118.70) than solo (119.75). However, its average idle time (91.40) is longer than solo's average idle time (50.10). As before, both MP2 and MP3 recorded better work efficiency when there were six agents in the team.MP3 performed best in terms of average work efficiency because it was able to capitalise on the positive pairing to work on tasks with higher complexities than the pairs' capablities.

Table 5.24 Five a six agents with evenly distributed capabilities working on complex tasks

Strategy	# of Agents	ACT	AET	AIT	WE
Solo	5	138.00	581.30	65.30	2.03
	6	101.50	537.40	34.90	1.79
	Average	**119.75**	**559.35**	**50.10**	**1.91**
MP1	5	CNC	CNC	CNC	CNC
	6	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
IP1	5	CNC	CNC	CNC	CNC
	6	CNC	CNC	CNC	CNC
	Average	CNC	CNC	CNC	CNC
MP2	5	160.50	482.80	300.70	2.36
	6	94.40	499.80	46.20	1.67
	Average	**127.45**	**491.30**	**173.45**	**2.02**
IP2	5	143.40	433.60	264.40	2.10
	6	89.50	446.40	69.00	1.58
	Average	**116.45**	**440.00**	**166.70**	**1.84**
MP3	5	141.60	533.60	158.40	2.08
	6	95.80	532.40	24.40	1.69
	Average	**118.70**	**533.00**	**91.40**	**1.89**
IP3	5	126.30	558.70	36.40	1.86
	6	96.10	530.70	16.30	1.70
	Average	**111.20**	**544.70**	**26.35**	**1.78**

Figure 5.24 Comparison of strategies for five a six agents with evenly distributed capabilities working on complex tasks



(a) (b)

5.2.25 Summary of the fixed testing

The summary result for the fixed environment are shown in Table 5.25. First of all, we can observe that the intelligent stategies (IP1, IP2 and IP3) performed well in almost all environments with the exception of three environments (six or more novices with Test Case 1, Test Case 2 and Test Case 6). It can also be observed that solo strategy performed better than must pair strategies but never better than the intelligent strategies. These results also suggest solo strategy can also be used in an enviornment where there are many expert developers working on easy tasks. The must pair strategies only work well when there are even number of agents in a team in an environment where there are more novice agents working on many complex tasks.

Table 5.25 Results summary for fixed environments

Efficiency compare with various teams and tasks	Test case 1 (10:20:20) Complex	Test case 2 (20:10:20) complex	Test case 3 (20:20:10) intermediate	Test case 4 (30:10:10) easy	Test case 5 (10:30:10) intermediate	Test case 6 (10:10:30) complex
Five more novices 3:1:1	IP2(1.86) IP3(2.10) MP2(2.24)	IP2(1.94) IP3(2.13) MP2(2.36)	IP3(1.95) IP2(1.96) MP3(2.45)	IP2(1.65) IP1(1.66) IP3(1.78)	IP2(1.79) IP3(1.84) IP1(1.92)	IP2(2.18) IP3(2.32) MP2(2.34)
Five intermediates 1:3:1	IP3(1.65) SOLO(1.73) IP2(1.81)	IP3(1.71) SOLO(1.78) IP2(1.90)	IP3(1.59) SOLO(1.68) IP2(1.73)	IP1(1.37) IP2(1.39) IP3(1.40)	IP2(1.34) IP1(1.35) IP3(1.43)	IP3(1.70) SOLO(1.72) MP3(2.06)
Five more experts 1:1:3	IP1(1.30) IP2(1.30) SOLO(1.38)	IP1(1.28) IP2(1.31) IP3(1.34)	IP1(1.29) IP2(1.32) SOLO(1.37)	IP1(1.27) IP2(1.28) IP3(1.31)	IP1(1.25) IP2(1.25) IP3(1.28)	IP2(1.26) IP2(1.27) IP3(1.31)
Five evenly distributed 2:2:1	IP3(1.75) IP2(1.82) SOLO(1.93)	IP3(1.78) IP2(1.93) SOLO(1.94)	IP3(1.74) IP2(1.76) SOLO(1.85)	IP1(1.43) IP2(1.50) IP3(1.58)	IP1(1.37) IP2(1.45) IP3(1.55)	IP3(1.86) IP2(2.10) SOLO(2.03)
Six more novices 3:2:1	MP3(1.95) IP3(2.10) IP2 ,MP2(2.16)	MP3(2.03) IP3(2.07) MP2(2.29)	IP3(1.93) IP2(1.97) MP3(2.05)	IP1(1.57) IP2(1.65), MP1(1.70)	IP2(1.64) MP2(1.72) MP3(1.78)	MP3(1.97) IP3(2.09) MP2(2.52)
Six more intermediates 1:4:1	IP3(1.68) SOLO(1.72) MP3(1.77)	IP3(1.71) SOLO(1.76) MP3(1.85)	IP3(1.67) SOLO(1.74) IP2(1.81)	IP1(1.44) IP2(1.46) SOLO(1.47)	IP1(1.40) IP2 (1.43) IP3(1.45)	IP3(1.70) SOLO(1.73) MP3(1.77)
Six more experts 1:2:3	IP1(1.30) IP2(1.34) IP3(1.43)	IP1(1.31) IP2(1.36) SOLO(1.45)	IP1(1.35) IP2(1.39) SOLO(1.46)	IP1(1.29) IP2(1.31) IP3(1.36)	IP1(1.28) IP2(1.29) IP3(1.36)	IP1(1.31) IP2(1.34) IP3(1.42)
Six evenly distributed 2:2:2	IP2(1.54) MP2(1.67) IP3(1.67)	IP2(1.58) IP3(1.62) MP3(1.75)	IP2(1.57) IP3(1.64) SOLO(1.77)	IP1(1.40) IP2(1.47) SOLO,IP3(1.56)	IP1(1.34) IP2(1.43) IP3(1.52)	IP2(1.58) MP2(1.67) MP3(1.69)

Table 5.26 Results summary for five and six agents

Agent/task	Evenly distributed tasks	Easy tasks	Intermediate tasks	Complex tasks
Five or six novice agents	IP2 (2.04) IP3(2.05) MP3(2.20)	IP1(1.62) IP2(1.65) IP3(1.77)	IP2(1.72) IP3(1.83) IP1(1.86)	MP3(2.16) IP3(2.21) IP2(2.36)
Five or six intermediate agents	IP3(1.67) SOLO(1.74) IP2(1.93)	IP1(1.41) IP2(1.43) IP3(1.44)	IP1(1.38) IP2(1.39) IP3(1.44)	IP3(1.70) SOLO(1.73) MP3(1.92)
Five or six expert agents	IP1(1.31) IP2(1.34) SOLO(1.42)	IP1(1.28) IP2(1.30) Ip3(1.34)	IP1(1.27) IP2(1.27) IP3(1.32)	IP1(1.29) IP2(1.30) IP3(1.37)
Five or six evenly distributed agents	IP2(1.70) IP3(1.70) SOLO(1.86)	IP1(1.43) IP2(1.50) IP3(1.58)	IP1(1.37) IP2(1.45) IP3(1.55)	IP3(1.78) IP2(1.84) MP3(1.89)

These results can be further summarised into the 16 generalised environments as shown in Table 5.26. Here, it can be observed that in all these environments, voluntary pairing worked best except when there were a majority of novice agents working on complex tasks. As discussed, in this situation, MP3 performed better than IP3 in terms of work efficiency as there were more positive pairings taking place. It can also be seen that, when most of the tasks have higher complexities than the team capabilities (i.e. more intermediate agents working on more complex tasks, more novice agents working on more intermediate tasks), IP3 worked best as it allowed a pair of agent to work on tasks with complexities higher than the pair's capability.

5.3 Random Testing

Table 5.27 and Figure 5.25 shows the result when running the seven strategies and the adaptive strategy defined in Chapter 3. It can be seen that in this envrionment, only Solo, MP3, IP3 and the adaptive strategies were able to complete all the tasks (labelled as 100 runs). IP1 and MP1 only managed to complete 48 runs out of 100, IP2 and MP2 were able to complete 93 ouf 100 runs. This means that Type 1 and Type 2 strategies can only work in some situations, whereas Type 3 strategies can be used for all situations. Similarly, the adaptive strategies work for all situations as it was tuned to select the best strategy to used based on the current environment. The average person hours (APH) is added to the table to show the total hours spent by the team.

Table 5.27 Random testing results

Strategy	ACT	AET	AIT	APH	Number of successful run	Average WE
Solo	99.32	431.30	63.39	542.20	100	1.97
MP1	94.98	410.93	84.39	517.59	48	1.89
IP1	76.58	333.81	64.45	421.66	48	1.53
MP2	105.99	442.68	109.64	573.55	93	2.08
IP2	86.63	377.19	70.62	472.60	93	1.72
MP3	107.60	466.58	94.97	582.65	100	2.12
IP3	89.86	425.40	32.46	490.69	100	1.78
Adaptive	89.77	372.48	90.48	489.95	100	1.77

As IP1, IP2, MP1 and MP2 were not able to complete all the runs successfully, we will only compare the four strategies that have completed all runs successfully (Solo, IP3, MP3 and Adaptive). As expected, the adaptive strategy recorded the best average work efficiency followed by IP3, solo and MP3. MP3 was worse off than solo due to the negative pairings that might have taken place. The performance of MP3 is also dependent on whether the number of agents in the team is even or odd. Adaptive strategy also recorded the shortest average completion time, the lowest average effort time, and the lowest average person hours. If this is translated to the cost of paying individual person for their hours worked on the sprint, both the adaptive strategy and IP3 will have the least cost. IP3 performed very closely to the adaptive strategy and recorded the lowest average idle time. This means, in the absence of an adaptive strategy, IP3 can be used as its performance is acceptable. Figure 5.25 shows the comparisons of the four strategies based on average completion time, average idle time, average effort time and person hour.

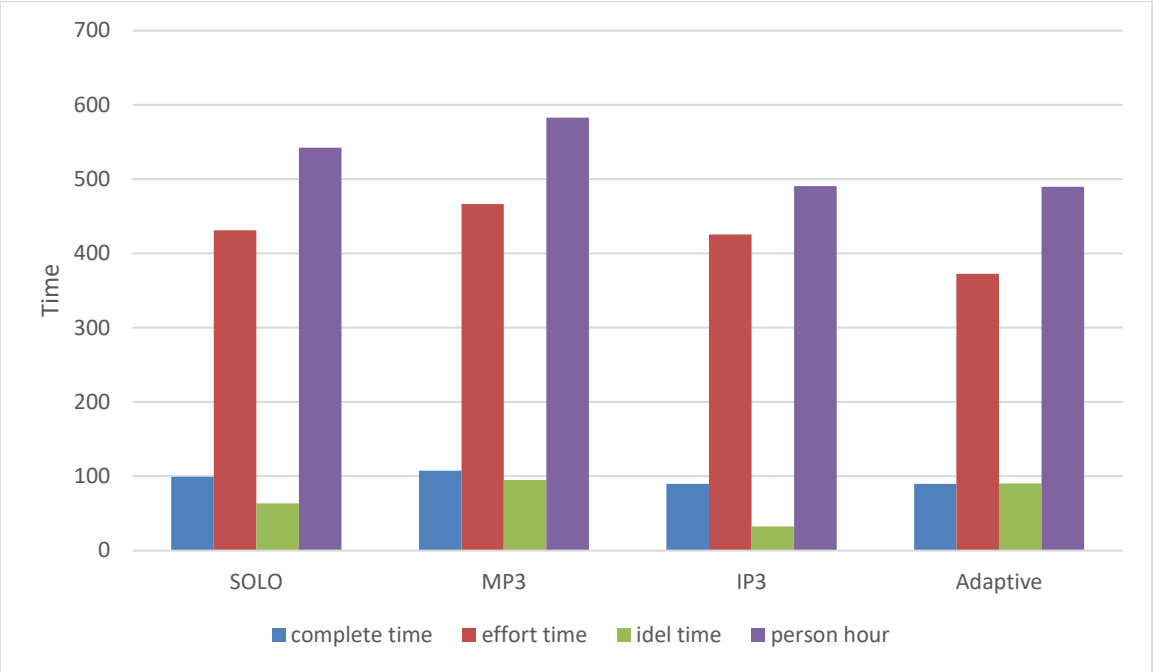


Figure 5.25 Comparison of strategies for random

Based on this results, both the adaptive strategy and the IP3 performed well in this environment. Adaptive strategy performed best as it changes its strategy of IP1, IP2 and IP3 depending on the type of environment it is in. This strategy is useful and can be used only if the environment can be accurately identified at the beginning of a software development project. However, in a real world setting this is not always true as the user requirements are dynamics and the categorisation of the task complexities and the agent caabilities may not be accurate. An alternative to this adaptive strategy is IP3 as it provides some level of flexibilities for pair with lower capabilities to work on task with higher level of complexities (with some penalties). This result addresses RQ4 on whether pairing is advantageous compared with not pairing. The result obtained suggests that pairing should only take place when it results in a positive pairing and when this is not possible, it is not necessary to pair.

5.4 Conflict Modelling

This experiment addresses RQ4. In this experiment, we chose to model conflict within the pair in the team using IP3 strategy as this strategy has very similar performance to adaptive strategy. Moreover, as discussed, the adaptive strategy only works if the environment can be accurately identified. We varied the rate of conflict from 0% to 20% by increments of 5%. In this experiment, we are keen to investigate how conflicts may affect the performance of the strategy (if this strategy is to be used in a real world setting). It can be seen from Table 5.28 and Figure 5.26 that as the rate of conflict increased from 0% to 20%, there is little effect of the work efficiency. There is a drop in the work efficiency from 1.79 to 1.81, which is a 0.02% decrease in work efficiency. This simply means that even though there

are conflicts within the pairs in the team, the performance of the strategy is stable and even when conflict is increased to 20%. Figure 5.26 compares the performance of IP3 for varying conflict rates.

Table 5.28 IP 3 conflict modelling results

Conflict rate	ACT	AEH	AIT	Person hours	WE
0	90.62	426.32	31.35	494.69	1.79
5	90.98	430.40	32.00	496.73	1.80
10	90.99	429.10	34.48	496.25	1.80
15	92.39	432.41	34.48	504.42	1.83
20	90.97	434.36	30.03	498.94	1.81

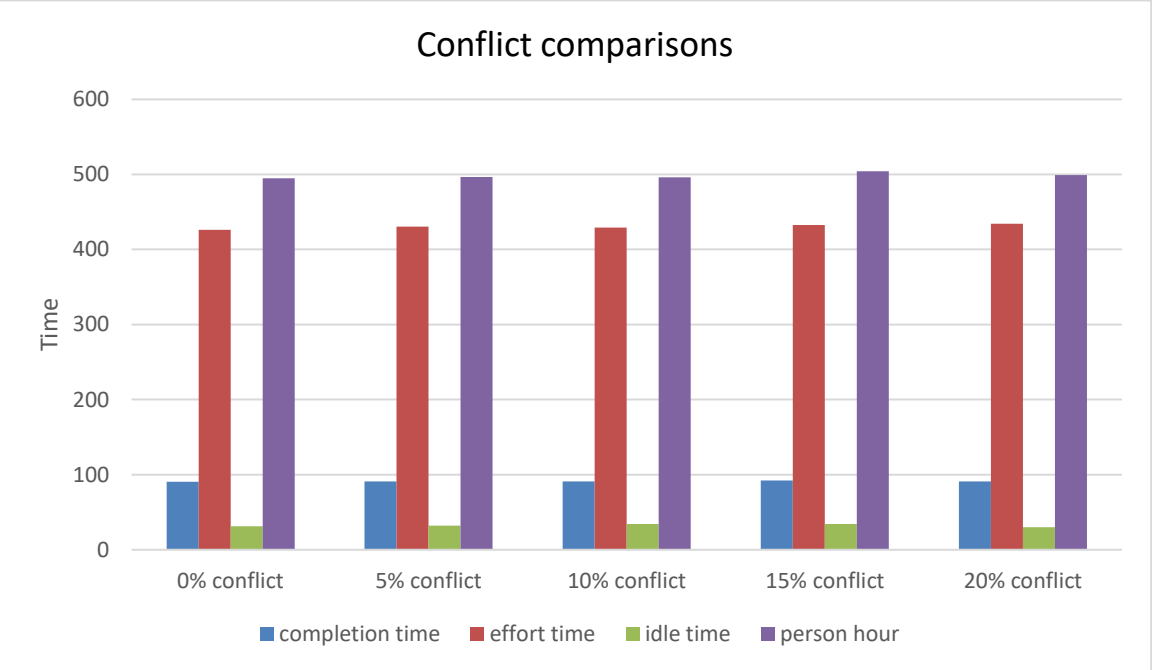


Figure 5.26 Comparison of strategies for conflict comparions

5.5Multi-sprint Modelling

This experiment addresses RQ4. The result obtained when implementing the strategy in multiple sprints for varying sprint durations is shown in Table 5.29 and Figure 5.27. It can be seen that the highest work efficiency (1.81) is recorded when the sprint duration is set to 100 . The lowest person hour is also observed when the sprint duration is 100. The lowest work efficiency is recorded when the sprint duration is 25. This is as expected since a shorter sprint duration means that complex tasks with

larger size may not be able to be completed within the short sprint duration. In contrast, longer sprint duration allows for more tasks to be worked in a single sprint.

Table 5.29 IP 3 multi-sprint modelling results

Sprint time	ACT	AET	AIT	Person hours	WE
25	100.51	411.52	107.60	548.64	1.99
50	94.17	419.56	59.59	514.20	1.87
75	92.20	423.56	51.29	502.87	1.83
100	91.27	427.42	38.41	498.41	1.81

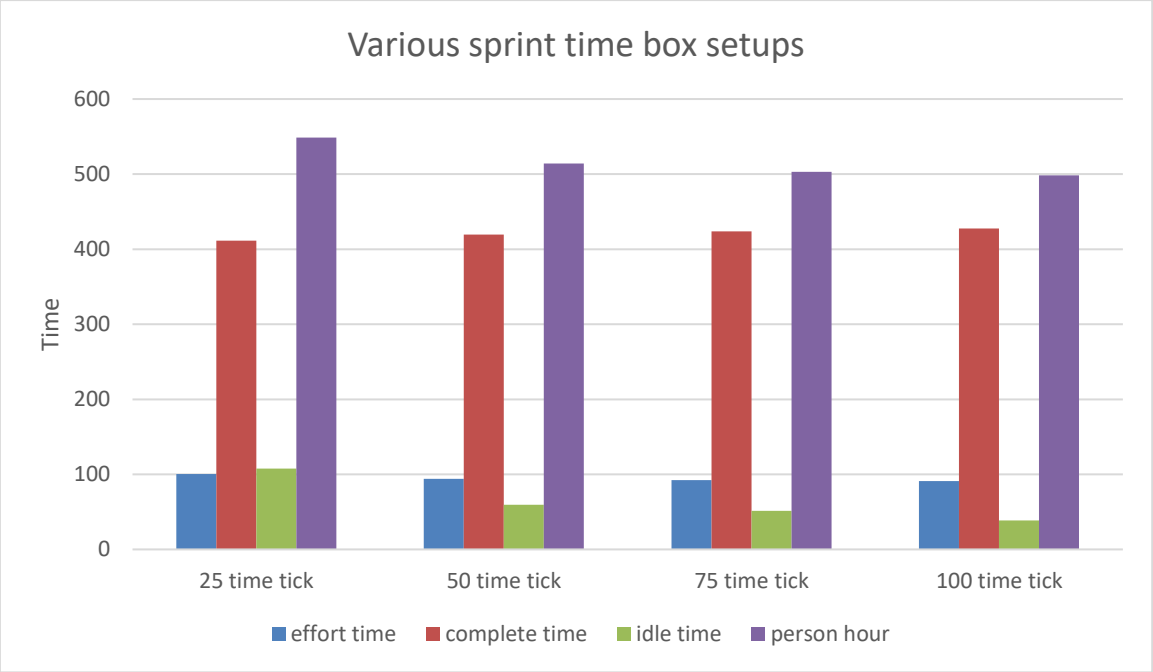


Figure 5.27 Comparison of strategies for various sprint time box

Similarly, the person hours for each sprint duration (as shown in Figure 5.27) is the lowest using sprint duration of 100. There is a difference of 4.46 (0.9%) in person hour with the sprint duration of 75. If the team needs more conversation with the product owner, having shorter sprint such as 75 (3 weeks) would be beneficial.

5.6 Summary

In this chapter we conducted experiments in a variety of settings to investigate the impact of pairing on Scrum software development project. We evaluated six pairing strategies and one solo strategy in these settings to determine which strategy works best for which environment. We also evaluated these strategies in a random environment to mimic the real world situation. We also investigated whether the performance of the selected strategy is affected when there are conflicst within the pair in the team. Finally, we implemented the selected strategy in a multi-sprint setting to determine its performance with varying sprint durations.

The fixed experiments which were run in all 48 environments address the following research questions RQ1, RQ2 and RQ3.

1. How does the adoption of pair programming impact the effort required to implement a software project?

2. What impact does the chosen pairing scheme have on the effort required?

3. How does team composition impact project completion under different pairing schemes?

The random experiments, conflict modelling and multi-sprint modelling address

4. In which situations is pairing advantageous compared with not pairing?

In summary the results obtained in these experiments successfully answered these research questions. In addition, the experiment provides insights on the what kind of strategy than can be used based the type of pairing, team composition and the task complexity. The summary, conclusions and the future work are discussed in the next chapter.

Chapter 6 Conclusions and Future Work

This chapter provides conclusions about what has been researched in the thesis, specifically, a summary of the research questions, what has been tried and solved, what has been achieved and the future work. We have developed a multi-agent system to simulate scrum team dynamics and to test 7 strategies which comprise of a solo strategy, 3 must pair strategies and 3 intelligent pair strategies. We conducted experiments in various settings including fixed testing, random testing, conflict modelling and multi-sprint modelling.

6.1 Summaries of the Chapters

Chapter 1
This chapter provided the background of agile approach with particular focus on Scrum and pair programming. It also provides the motivation, justification and the main objective of this work.

Chapter 2
This chapter reviewed related literatures with respect to the projects which include software development model, scrum model, pair programming and software process modelling and simulation. The research gaps and the research questions were also discussed.

Chapter 3

This chapter discussed three type of strategies: solo, intelligent pair and must pair. The strategies were designed based the pair programming literature review to guarantee its correctness. These three types of strategies had their own sub-types, which provided further details on how a type of strategy can have multiple purposes based on different design motivations. There were seven strategies designed for testing that were used to investigate the impact of pairing in a Scrum software development project.

Chapter 4

Chapter 4 presented the design and implementation of an agent-based model to simulate and evaluate strategies for the Scrum software development process. This simulation was implemented based the Java Agent Development framework (JADE) where the agents were able to interact and exchange information with one another using FIPA-ACL communication language. The multi-agent system simulation is described in detail.

Chapter 5

This chapter included the realization of large numbers of experiments designed in Chapter 3 to test the seven strategies for four types of experiments (fixed type, random type, conflict type and multi-sprint type). This chapter provided data to answer to our research questions and draw conclusions for those strategies.

6.2 Research Findings

The research questions that were addressed during this study are as follows.

1. **How does the adoption of pair programming impact the effort required to implement a software project?**

   Yes, the effort is impacted by adopting pair programming. Pair programming is used in both intelligent pair and must pair, however in the intelligent pair, only positive pairing is considered.

   In order to evaluate the impact of pair programming (either MP or IP), we compared the performance of solo programming with pair programming, and we found that must pair is not always better than solo, but intelligent pair is always better than solo. Based on the result obtained, the must pair only performed well in one of the 48 environments that were tested. In contrast, the intelligent pair performed well in 47 environments.

   Moreover, the average effort time recorded by the intelligent pair are lower than that of the solo strategy. The use of intelligent strategy also suggests that pairing is recommended when it results in a negative pairing and when no positive pairing is possible, solo programming is recommended to reduce the team's average idle time.

2. **What impact does the pairing scheme chosen have on the effort required?**

   The strategy chosen will impact the efficiency of team working performance. In RQ1, we show that pairing was better than solo (except when there was negative pairing). Then we investigated the pairing schemes (MP and IP) in more detail and what kind of tasks were taken by pairs (Type 1, Type 2 and Type 3).

   There are seven strategies designed and implemented, intelligent pair (IP1, IP2 and IP3), must pair (MP1, MP2 and MP3)) and solo. Type 1 strategy dictates that a pair of agent can only work on a task that has the same complexity as the pair capability. Type 2 allows a pair of agent to work on task in the same category of complexity as the pair capability (i.e. novice working on easy task,

intermediate working on intermediate task and expert working on complex task). Type 3 is the most flexible strategy as it allows a pair of agent to work on task in a category higher than the pair capability (novice working on intermediate task, intermediate working on complex task) with a penalty.

In the must pair strategy, negative pairing is allowed as all tasks must be worked by a pair of agents. As such, it was found that must pair strategy only worked well when there are even number of members in the team.

Intelligent pair performed well in most of the environments because it avoids negative pairing. This strategy worked for both odd numbered and even numbered team.

Solo strategy is used to compare with must pair strategy and intelligent pair strategy. Solo strategy will be used when positive pairing is not possible and when the task to be completed has the same complexity as the agent capability.

Based on the summary of the results shown in Table 3.10, it can be seen that the intelligent strategies (IP1, IP2 and IP3) performed well in almost all environments.

3. How does team composition impact project completion under different pairing schemes?

The analysis for varying team composition and varying task complexities are summarised in Table 5.26. There are four types of team; team with novice agents as the majority, team with intermediate agents as majority, team with expert agents as majority and team with evenly distributed capability. Based on this result, we can conclude that the intelligent pair performed consistently well in all the generalised environment except for one (a team with novice agents as majority working on complex tasks). The completion time for all intelligent pairs were also shorter when compared to the must pair strategies and solo strategy. In this setting, MP3 did well because most of the pairings were positive pairing.

4. In what situations is pairing advantageous compared with not pairing?

Following on from RQ3 when, after analysing all possible environments and identifying the best strategy to be used for each environment, we tested all the 7 strategies and an adaptive strategy in a random environment. The adaptive strategy is introduced to investigate what happen if we

tuned the strategy to the prevailing environment. Using adaptive strategy, we assumed that the environment can be accurately identified and hence we can select the best strategy that can be used for this environment.

Based on the result, we can conclude that the adaptive strategy is the best strategy and should be used if we can accurately identify the environment it is in. However, when this is not possible (as requirement changes dynamically and we may not be able to accurately categorise the pair capability and the task complexity), IP3 can be used as an alternative strategy. This is because, the performance of IP3 (in terms of work efficiency, average completion time and person hour) is very similar to the adaptive strategy. This result strongly suggests that pairing is advantageous when it is a positive pairing and when no positive pairing is possible, solo programming is preferred. We also investigated if this kind of pairing is resilient to potential conflicts within the pairs in the team. We tested IP3 in a random environment with conflict rates of 5%, 10%, 15% and 20%. The result in Table 5.28 shows that there is a minimal effect of the work efficiency when we increase the conflict rate from 0% to 20%. Even though the conflict rate was increased to 20%, the work efficiency only decreased from 1.79 to 181 which is a decrease of 0.02%.

The experiments for the fixed and random environments were conducted based on a single sprint. We also investigated whether the performance of IP3 was affected by running it in multiple sprints using varying time boxes. In this experiment, we tested the performance of IP3 using sprint duration of 1 week, 2 weeks, 3 weeks and 4 weeks. Based on Table 5.29, it can be seen that the highest work efficiency (1.81) is recorded when the sprint duration is set to 100. The lowest person hour is also observed when the sprint duration is 100. The lowest work efficiency is recorded when the sprint duration is 25. The result of this experiment seems to suggest that the larger the time box, the better the efficiency and the shorter the completion time.

6.3 Research Contribution

A comprehensive investigation was undertaken into pair programming in a Scrum team by providing an advanced simulation tool that was developed using a multi-agent system (FIPA standard). Seven strategies were designed and provided to compare the performance of various kinds of pair and solo programming. The concept of negative pairing, positive pairing, cross level pairing and same level pairing were all considered in the design of these strategies. Experiments were undertaken to test those strategies under various contexts, such as fixed and random contexts, conflict modelling and multi-sprint modelling. The intelligent pair algorithm was observed to give the best performance in all contexts and to provide stability in real world simulations. The adaptive strategy also performed well

in the random environment where it was able to change its strategy based on the prevailing environment and would work well if the environment can be accurately identified.

This research provides a comprehensive analysis of different kind of pairings; must pair and voluntary pair and suggests how pairing should be undertaken based on varying situations (varying agent capabilities and varying task complexities).

A complete approach on how to undertake multi-agent modelling on Scrum teams through strategy comparisons was also provided. This can be further enhanced to develop many different types of Scrum team working strategies and evaluate them in the various environments.

This research has fulfilled the research gaps described in Chapter 2, in that it undertook a thorough investigation in pair programming strategies within a Scrum team. Based on the results obtained, Intelligent Pair Type 3 strategy and the adaptive strategy can be used in a real world setting as it was shown that these two strategies were able to obtain the highest work efficiency irrespective of the team composition and task complexity.

6.4 Future Work

The current simulation models developer agents with a single capability value by giving it a value between 1 – 10. A future extension to this would be providing agents with per skill capabilities and describing task complexity in terms of skills required to complete. This would allow the exploration of specialist agents versus representations of cross-functionally skilled team members.

A further extension to the developer agents, would see that addition of personality characteristics to allow the modelling of the interaction of differing personality types when pairs are formed.

Team members in a development team, over time, increase their skills and capabilities through external learning and also by working on tasks within projects, currently this within project learning is not modelled in the simulation and would be an addition that would help provide insight into how teams evolve overtime, and the impact that pairing has on learning rates as junior developers are able to tackle more complex tasks alongside more senior developers increasing their exposure to complex problems.

Conflict modelling in the current study has been limited to a percentage of tasks “failing” and requiring additional work. Further work in this area would add nuances around the reasons for tasks “failing” or requiring extra work. This would build on the additions outlined above, in particular the personality additions to developers, perhaps limiting the pairs that could work together following a conflict.

References

Abar, S., Theodoropoulos, G. K., Lemarinier, P., & O'Hare, G. M. P. (2017). Agent Based Modelling and Simulation tools: A review of the state-of-art software. *Computer Science Review, 24*, 13-33. doi:https://doi.org/10.1016/j.cosrev.2017.03.001

Agarwal, R. (2007). *A flexible model for multi-agent based simulation of software development process*. Auburn University.

Agarwal, R., & Umphress, D. (2010). *A flexible model for simulation of software development process*. Paper presented at the meeting of the Proceedings of the 48th Annual Southeast Regional Conference, Oxford, Mississippi. doi:10.1145/1900008.1900064

Ali, N., Petersen, K., & Wohlin, C. (2014). A systematic literature review on the industrial use of software process simulation. *J. Syst. Softw., 97*(C), 65-85. doi:10.1016/j.jss.2014.06.059

Ali, S., Ali, H. H., Qayyum, S., Sohail, F., Tahir, F., Maqsood, S., & Adil, M. (2019). Multi-agent System Using Scrum Methodology for Software Process Management*Springer Singapore.* Singapore. Abstract retrieved from 10.1007/978-981-13-6052-7_68

Alsalemi, A. M., & Yeoh, E. (2015, 16-17 Dec. 2015). A survey on product backlog change management and requirement traceability in agile (Scrum) Symposium conducted at the meeting of the 2015 9th Malaysian Software Engineering Conference (MySEC) doi:10.1109/MySEC.2015.7475219

Anderson, D. J., Concas, G., Lunesu, M. I., Marchesi, M., & Zhang, H. (2012). A Comparative Study of Scrum and Kanban Approaches on a Real Case Study Using Simulation*Springer Berlin Heidelberg.* Berlin, Heidelberg. Abstract retrieved from 10.1007/978-3-642-30350-0_9

Andrejczuk, E. D. (2018). *Artificial intelligence methods to support people management in organisations*.

Argote, L., & Fahrenkopf, E. (2016). Knowledge transfer in organizations: The roles of members, tasks, tools, and networks. *Organizational Behavior and Human Decision Processes, 136*, 146-159. doi:https://doi.org/10.1016/j.obhdp.2016.08.003

Arisholm, E., Gallis, H., Dyba, T., & Sjoberg, D. I. K. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Transactions on Software Engineering, 33*(2), 65-86. doi:10.1109/TSE.2007.17

Bassi, A. (2016). *Scrum Sim - A Simulation Game to Learn the Scrum Agile Framework*. Harrisburg University of Science and Technology. Retrieved from Retrieved from https://digitalcommons.harrisburgu.edu/pmgt_dandt/11

Beck, K., Grenning, J., & Martin, R. (2020, 2020). *Manifesto for Agile Software Development*. Retrieved from https://agilemanifesto.org/.

Bellifemine, F., Caire, G., & Greenwood, D. (2007). *Develpoing Multi-Agent Systems with JADE*. England: Wiley Series in Agent Technology.

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences, 99*(suppl 3), 7280-7287. doi:10.1073/pnas.082080899

Boulahbel-Bachari, S., & El Saadi, N. (2019). Migration and Self-selection: An Agent-Based Model*Springer International Publishing.* Cham. Abstract retrieved from 10.1007/978-3-030-31362-3_28

Cao, L., Ramesh, B., & Abdel-Hamid, T. (2010). Modeling dynamics in agile software development. *ACM Trans. Manage. Inf. Syst., 1*(1), 1-26. doi:10.1145/1877725.1877730

Çetin, E., & Durdu, P. (2019). Blended Scrum model for software development organizations. *Journal of Software: Evolution and Process, 31*(2), e2147. doi:10.1002/smr.2147

Cherif, R., & Davidsson, P. (2010). Software Development Process Simulation: Multi Agent-Based Simulation versus System Dynamics*Springer Berlin Heidelberg.* Berlin, Heidelberg. Abstract retrieved from 10.1007/978-3-642-13553-8_7

Cocco, L., Mannaro, K., Concas, G., & Marchesi, M. (2011). Simulating Kanban and Scrum vs. Waterfall with System Dynamics*Springer Berlin Heidelberg.* Berlin, Heidelberg. Abstract retrieved from 10.1007/978-3-642-20677-1_9

Cockburn, A., & Williams, L. (2001). The costs and benefits of pair programming. In *Extreme*

programming examined (pp. 223-243): Addison-Wesley Longman Publishing Co., Inc.
Coman, I. D., Robillard, P. N., Sillitti, A., & Succi, G. (2014). Cooperation, collaboration and pair-programming: Field studies on backup behavior. *Journal of Systems and Software, 91*, 124-134. doi:https://doi.org/10.1016/j.jss.2013.12.037
D'Angelo, S., & Begel, A. (2017). *Improving Communication Between Pair Programmers Using Shared Gaze Awareness*. Paper presented at the meeting of the Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, Colorado, USA. doi:10.1145/3025453.3025573
da Silva, F. Q. B., França, A. C. C., Suassuna, M., de Sousa Mariz, L. M. R., Rossiley, I., de Miranda, R. C. G., . . . Espindola, E. (2013). Team building criteria in software projects: A mix-method replicated study. *Information and Software Technology, 55*(7), 1316-1340. doi:https://doi.org/10.1016/j.infsof.2012.11.006
de O. Melo, C., S. Cruzes, D., Kon, F., & Conradi, R. (2013). Interpretative case studies on agile team productivity and management. *Information and Software Technology, 55*(2), 412-427. doi:https://doi.org/10.1016/j.infsof.2012.09.004
Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software, 85*(6), 1213-1221. doi:https://doi.org/10.1016/j.jss.2012.02.033
Dybå, T., Arisholm, E., Sjøberg, D. I. K., Hannay, J. E., & Shull, F. (2007). Are Two Heads Better than One? On the Effectiveness of Pair Programming. *IEEE Software, 24*(6), 12-15. doi:10.1109/MS.2007.158
Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology, 50*(9), 833-859. doi:https://doi.org/10.1016/j.infsof.2008.01.006
Farid, W. M., & Mitropoulos, F. J. (2013, 4-7 April 2013). NORPLAN: Non-functional Requirements Planning for agile processes Symposium conducted at the meeting of the 2013 Proceedings of IEEE Southeastcon doi:10.1109/SECON.2013.6567463
Ghimire, D., Gibbs, S., & Charters, S. (2016, 11th-13th July 2016). *Software Development Team Views of Success Factors*

in Agile projects Paper presented at the meeting of the the 7th annual conference of Computing and Information Technology Research and Education

New Zealand (CITRENZ2016) Wellington, New Zealand, .
Gilbert, N., & Troitzsch, K. G. (2005). *Simulation for the Social Scientist*: Open University Press.
Griffith, I., Taffahi, H., Izurieta, C., & Claudio, D. (2014, 7-10 Dec. 2014). A simulation study of practical methods for technical debt management in agile software development Symposium conducted at the meeting of the Proceedings of the Winter Simulation Conference 2014 doi:10.1109/WSC.2014.7019961
Hagemeister, M., & Rodríguez-Castellanos, A. (2019). Knowledge acquisition, training, and the firm's performance: A theoretical model of the role of knowledge integration and knowledge options. *European Research on Management and Business Economics, 25*(2), 48-53. doi:https://doi.org/10.1016/j.iedeen.2019.02.003
Haider, M., & Ali, I. (2011). Evaluation of the Effects of Pair Programming on Performance and Social Practices in Distributed Software Development
Han, J., Liu, J., & Lei, H. (2019). *Agent-Based Simulation of Emergency Response of Urban Oil and Gas Pipeline Leakage*. Paper presented at the meeting of the Proceedings of the 11th International Conference on Computer Modeling and Simulation, North Rockhampton, QLD, Australia. doi:10.1145/3307363.3307396
Hanakawa, N., Matsumoto, K.-i., & Torii, K. (2002). A Knowledge-Based Software Process Simulation Model [journal article]. *Annals of Software Engineering, 14*(1), 383-406. doi:10.1023/a:1020574228799
Hannay, J., Dyb, T., Arisholm, E., & Sjobery, D. (2009). The effectiveness of pair programming: A meta-analysis. *Inf. Softw. Technol., 51*(7), 1110-1122. doi:10.1016/j.infsof.2009.02.001
Hashimoto, A., Takata, K., Ito, N., Matoba, R., Tani, K., & Maeda, Y. (2019). *Study of the Influence of an*

Obstacle on the Evacuation Behavior Using Multi-Agent Simulation where the Intimate Space around each Agent is Considered. Paper presented at the meeting of the Proceedings of the 11th International Conference on Computer Modeling and Simulation, North Rockhampton, QLD, Australia. doi:10.1145/3307363.3307369

Hoda, R., & Murugesan, L. K. (2016). Multi-level agile project management challenges: A self-organizing team perspective. *Journal of Systems and Software, 117*, 245-257. doi:https://doi.org/10.1016/j.jss.2016.02.049

Ivanov, D., Kapustyan, S., Kalyaev, A., & Korovin, I. (2019). Decision Support Systems for the Oil Fields with Cloud Multiagent Service*Springer International Publishing.* Cham. Abstract retrieved from 10.1007/978-3-030-31362-3_3

Jennings, N. R., & Wooldridge, M. J. (1998). *Agent Technology:Foundations, Applications, and Markets* (Vol. 1): Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-662-03678-5

Joslin, D., & Poole, W. (2005, 4-4 Dec. 2005). Agent-based simulation for software project planning Symposium conducted at the meeting of the Proceedings of the Winter Simulation Conference, 2005. doi:10.1109/WSC.2005.1574359

Justice, J. (2018). *The 3-5-3 of Scrum*. Retrieved from https://www.scruminc.com/the-3-5-3-of-Scrum/.

Kellner, M. I., Madachy, R. J., & Raffo, D. M. (1999). Software process simulation modeling: Why? What? How? *Journal of Systems and Software, 46*(2), 91-105. doi:https://doi.org/10.1016/S0164-1212(99)00003-5

Khmelevsky, Y., Li, X., & Madnick, S. (2017, 24-27 April 2017). Software development using agile and scrum in distributed teams Symposium conducted at the meeting of the 2017 Annual IEEE International Systems Conference (SysCon) doi:10.1109/SYSCON.2017.7934766

Košinár, M., & Štrba, R. (2013). Simulations of Agile Software Processes for Healthcare Information Systems Development Based on Machine Learning Methods. *IFAC Proceedings Volumes, 46*(28), 175-180. doi:https://doi.org/10.3182/20130925-3-CZ-3023.00028

Lei, H., Ganjeizadeh, F., Jayachandran, P. K., & Ozcan, P. (2017). A statistical analysis of the effects of Scrum and Kanban on software development projects. *Robot. Comput.-Integr. Manuf., 43*(C), 59-67. doi:10.1016/j.rcim.2015.12.001

Licorish, S. A., & MacDonell, S. G. (2014). Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study. *Information and Software Technology, 56*(12), 1578-1596. doi:https://doi.org/10.1016/j.infsof.2014.02.004

Licorish, S. A., & MacDonell, S. G. (2017). Exploring software developers' work practices: Task differences, participation, engagement, and speed of task resolution. *Information & Management, 54*(3), 364-382. doi:https://doi.org/10.1016/j.im.2016.09.005

Licorish, S. A., & MacDonell, S. G. (2018). Exploring the links between software development task type, team attitudes and task completion performance: Insights from the Jazz repository. *Information and Software Technology, 97*, 10-25. doi:https://doi.org/10.1016/j.infsof.2017.12.005

Lin. (2013). *Context-aware task allocation for distributed agile team*. Paper presented at the meeting of the Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, Silicon Valley, CA, USA. doi:10.1109/ase.2013.6693151

LIN. (2015). Human Factors in Agile Software Development. *ArXiv, abs/1502.04170*.

Lin, Yu, H., & Shen, Z. (2014). An Empirical Analysis of Task Allocation in Scrum-based Agile Programming. *ArXiv, abs/1411.6201*.

Lin, Yu, H., Shen, Z., & Miao, C. (2014). *Studying task allocation decisions of novice agile teams with data from agile project management tools*. Paper presented at the meeting of the Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, Vasteras, Sweden. doi:10.1145/2642937.2642959

Lin, & Zhang, Q. (2019). Time scales of knowledge transfer with learning and forgetting. *Physica A: Statistical Mechanics and its Applications, 525*, 704-713. doi:https://doi.org/10.1016/j.physa.2019.03.084

Lindsjørn, Y., Sjøberg, D. I. K., Dingsøyr, T., Bergersen, G. R., & Dybå, T. (2016). Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software, 122*, 274-286. doi:https://doi.org/10.1016/j.jss.2016.09.028

López-Martínez, J., Juárez-Ramírez, R., Huertas, C., Jiménez, S., & Guerra-García, C. (2016, 27-29 April 2016). Problems in the Adoption of Agile-Scrum Methodologies: A Systematic Literature Review Symposium conducted at the meeting of the 2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT) doi:10.1109/CONISOFT.2016.30

Lui, K. M., & Chan, K. C. C. (2006). Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-Computer Studies, 64*(9), 915-925. doi:https://doi.org/10.1016/j.ijhcs.2006.04.010

Lunesu, M. I., Münch, J., Marchesi, M., & Kuhrmann, M. (2018). Using simulation for understanding and reproducing distributed software development processes in the cloud. *Information and Software Technology, 103*, 226-238. doi:https://doi.org/10.1016/j.infsof.2018.07.004

Macal, C. M., & North, M. J. (2006, 3-6 Dec. 2006). Tutorial on Agent-Based Modeling and Simulation PART 2: How to Model with Agents Symposium conducted at the meeting of the Proceedings of the 2006 Winter Simulation Conference doi:10.1109/WSC.2006.323040

Macal, C. M., & North, M. J. (2008). *Agent-based modeling and simulation: ABMS examples*. Paper presented at the meeting of the Proceedings of the 40th Conference on Winter Simulation, Miami, Florida.

Mahnič, V., & Hovelja, T. (2012). On using planning poker for estimating user stories. *Journal of Systems and Software, 85*(9), 2086-2095. doi:https://doi.org/10.1016/j.jss.2012.04.005

Masood, Z., Hoda, R., & Blincoe, K. (2017, 23-23 May 2017). Motivation for Self-Assignment: Factors Agile Software Developers Consider Symposium conducted at the meeting of the 2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE) doi:10.1109/CHASE.2017.18

Maxim, B. R., Kaur, R., Apzynski, C., Edwards, D., & Evans, E. (2016, 12-15 Oct. 2016). An agile software engineering process improvement game Symposium conducted at the meeting of the 2016 IEEE Frontiers in Education Conference (FIE) doi:10.1109/FIE.2016.7757682

Mehmood, S., Ahmed, S., & Kristensen, A. S. (2019). *Application of Integrated Model of Evacuation Psychology in an Agent-based Simulation*. Paper presented at the meeting of the Proceedings of the 11th International Conference on Computer Modeling and Simulation, North Rockhampton, QLD, Australia. doi:10.1145/3307363.3307389

Moe, N. B., Dings, T., #248, yr, Dyb, T., & #229. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project. *Inf. Softw. Technol., 52*(5), 480-491. doi:10.1016/j.infsof.2009.11.004

Moe, N. B., & Dingsøyr, T. (2008a). Scrum and Team Effectiveness: Theory and Practice*Springer Berlin Heidelberg.* Berlin, Heidelberg. Abstract retrieved from 10.1007/978-3-540-68255-4_2

Moe, N. B., & Dingsøyr, T. (2008b, 2008//). Scrum and Team Effectiveness: Theory and Practice. In P. Abrahamsson, R. Baskerville, K. Conboy, B. Fitzgerald, L. Morgan & X. Wang (Chair), *Springer Berlin Heidelberg.* Symposium conducted at the meeting of the Agile Processes in Software Engineering and Extreme Programming, Berlin, Heidelberg.

Müller, M. M. (2007). Do programmer pairs make different mistakes than solo programmers? *Journal of Systems and Software, 80*(9), 1460-1471. doi:https://doi.org/10.1016/j.jss.2006.10.032

Nadler, D., Hackman, J. R., & Lawler, E. E. (1979). *Managing organizational behavior*

Nicholas, R. J., & Michael, J. W. (1998). *Agent technology: foundations, applications, and markets*: Springer-Verlag New York, Inc.

Nilsson, K. (2003). *Increasing Quality with Pair Programming - An Investigation of Using Pair Programming as a Quality Tool*. Blekinge Institute of Technology Sweden. Retrieved from https://www.diva-portal.org/smash/get/diva2:831281/FULLTEXT01.pdf

Noori, F., & Kazemifard, M. (2015, 27-29 April 2015). Simulation of pair programming using multi-agent and MBTI personality model Symposium conducted at the meeting of the 2015 Sixth International Conference of Cognitive Science (ICCS) doi:10.1109/COGSCI.2015.7426665

Orłowski, C., Bach-Dąbrowska, I., Kapłański, P., & Wysocki, W. (2014). Hybrid Fuzzy-ontological Project Framework of a Team Work Simulation System. *Procedia Computer Science, 35*, 1175-1184. doi:https://doi.org/10.1016/j.procs.2014.08.214

Osama, A.-B., & James, M. (2015). The kanban approach, between agility and leanness: a systematic review. *Empirical Software Engineering, 20*(6), 1861-1897. doi:10.1007/s10664-014-9340-x

Palmquist, S., Lapham, M. A., Garcia-Miller, S., Chick, T., & Ozkaya, I. (2013). Parallel Worlds: Agile and Waterfall Differences and Similarities*Retrieved November 03, 2020, from the Software Engineering Institute, Carnegie Mellon University* Retrieved from http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=62901

Perkusich, M., Almeida, H. O. d., & Perkusich, A. (2013). *A model to detect problems on scrum-based software development projects*. Paper presented at the meeting of the Proceedings of the 28th Annual ACM Symposium on Applied Computing, Coimbra, Portugal. doi:10.1145/2480362.2480560

Phillips, J. (2006). *Team-RUP: An Agent-Based Simulation Study of Team Behavior In Software Development Organizations*. Auburn University Alabama. Retrieved from https://etd.auburn.edu/handle/10415/497

Plonka, L., Sharp, H., van der Linden, J., & Dittrich, Y. (2015). Knowledge transfer in pair programming: An in-depth analysis. *International Journal of Human-Computer Studies, 73*, 66-78. doi:https://doi.org/10.1016/j.ijhcs.2014.09.001

Quaglia, E. J., & Tocantins, C. A. (2011, 11-14 Dec. 2011). Simulation projects management using Scrum Symposium conducted at the meeting of the Proceedings of the 2011 Winter Simulation Conference (WSC) doi:10.1109/WSC.2011.6148038

Ramanujam, R., & Lee, I. (2011, 21-23 June 2011). Collaborative and competitive strategies for agile scrum development Symposium conducted at the meeting of the The 7th International Conference on Networked Computing and Advanced Information Management

Royce, W. W. (1987). *Managing the development of large software systems: concepts and techniques*. Paper presented at the meeting of the Proceedings of the 9th international conference on Software Engineering, Monterey, California, USA. Retrieved from https://dl.acm.org/doi/10.5555/41765.41801 doi:10.5555/41765.41801

Schwaber, k., & Sutherland, J. (2017). *The Scrum Guide*. Retrieved from https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.

Sheffield, J., & Lemétayer, J. (2013). Factors associated with the software development agility of successful projects. *International Journal of Project Management, 31*(3), 459-472. doi:https://doi.org/10.1016/j.ijproman.2012.09.011

Shiohama, R., Washizaki, H., Kuboaki, S., Sakamoto, K., & Fukazawa, Y. (2012, 13-17 Aug. 2012). Estimate of the Appropriate Iteration Length in Agile Development by Conducting Simulation Symposium conducted at the meeting of the 2012 Agile Conference doi:10.1109/Agile.2012.16

Silva, I. J. D., Rayadurgam, S., & Heimdahl, M. P. E. (2015). *A reference model for simulating agile processes*. Paper presented at the meeting of the Proceedings of the 2015 International Conference on Software and System Process, Tallinn, Estonia. doi:10.1145/2785592.2785615

Šmite, D., Moe, N. B., Šāblis, A., & Wohlin, C. (2017). Software teams and their knowledge networks in large-scale software development. *Information and Software Technology, 86*, 71-86. doi:https://doi.org/10.1016/j.infsof.2017.01.003

Sommerville, I. (2016). *Software Engineering*. England: PEARSON.

Song, H., Chien, A. T., Fisher, J., Martin, J., Peters, A. S., Hacker, K., . . . Singer, S. J. (2015). Development and Validation of the Primary Care Team Dynamics Survey. *Health Services Research, 50*(3), 897-921. doi:10.1111/1475-6773.12257

Spasic, B., & Onggo, B. S. S. (2012, 9-12 Dec. 2012). Agent-based simulation of the software development process: A case study at AVL Symposium conducted at the meeting of the Proceedings of the 2012 Winter Simulation Conference (WSC) doi:10.1109/WSC.2012.6465117

Stober, T., & Hansmann, U. (2010). Considerations on Teaming and Leadership. In *Agile Software Development: Best Practices for Large Software Development Projects* (pp. 75-92). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from https://doi.org/10.1007/978-3-540-70832-2_5. doi:10.1007/978-3-540-70832-2_5

Stray, V., Sjøberg, D. I. K., & Dybå, T. (2016). The daily stand-up meeting: A grounded theory study. *Journal of Systems and Software, 114*, 101-124. doi:https://doi.org/10.1016/j.jss.2016.01.004

Tamburri, D. A., Razo-Zapata, I. S., Fernández, H., & Tedeschi, C. (2012, 4-4 June 2012). Simulating awareness in global software engineering: A comparative analysis of Scrum and Agile Service Networks Symposium conducted at the meeting of the 2012 4th International Workshop on

Principles of Engineering Service-Oriented Systems (PESOS) doi:10.1109/PESOS.2012.6225933
Tyrychtr, J., Pelikán, M., Kvasnička, R., Ander, V., Benda, T., & Vrana, I. (2019). Multi-agent System in Smart Econometric Environment*Springer International Publishing.* Cham. Abstract retrieved from 10.1007/978-3-030-30329-7_38
Uzzafer, M. (2013). A simulation model for strategic management process of software projects. *Journal of Systems and Software, 86*(1), 21-37. doi:https://doi.org/10.1016/j.jss.2012.06.042
Vinod, V., Padmanabhuni, K., Tadiparthi, H. P., Yanamadala, M., & Madina, S. (2012). Effective Pair Programming Practice- An Experimental Study
Wang, X., Wang, J., & Zhang, R. (2019). The optimal feasible knowledge transfer path in a knowledge creation driven team. *Data & Knowledge Engineering, 119*, 105-122. doi:https://doi.org/10.1016/j.datak.2019.01.002
Wang, Z. (2019a). The Compare of Solo Programming Strategies in a Scrum Team: A Multi-agent Simulation Tool for Scrum Team Dynamics*Springer International Publishing.* Cham. Abstract retrieved from 10.1007/978-3-030-30329-7_32
Wang, Z. (2019b). *Estimating Productivity in a Scrum team: A Multi-Agent Simulation*. Paper presented at the meeting of the Proceedings of the 11th International Conference on Computer Modeling and Simulation, North Rockhampton, QLD, Australia. doi:10.1145/3307363.3310985
William, R., & Uri, W. (2015). An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo.
Williams, L. (2002). *Pair Programming Illuminated*: Addison-Wesley Longman Publishing Co., Inc.
Williams, L., & Kessler, R. (2001, 2001/01/01). *Experiments with Industry's "Pair-Programming" Model in the Computer Science Classroom*. Paper presented at the meeting of the Computer Science Education, Retrieved from https://doi.org/10.1076/csed.11.1.7.3846 doi:10.1076/csed.11.1.7.3846
Williams, L., McDowell, C., Nagappan, N., Fernald, J., & Werner, L. (2003, 30 Sept.-1 Oct. 2003). Building pair programming knowledge through a family of experiments Symposium conducted at the meeting of the 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings. doi:10.1109/ISESE.2003.1237973
Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education, 12*(3), 197-212. doi:10.1076/csed.12.3.197.8618
Wood, S., Michaelides, G., & Thomson, C. (2013). Successful extreme programming: Fidelity to the methodology or good teamworking? *Information and Software Technology, 55*(4), 660-672. doi:https://doi.org/10.1016/j.infsof.2012.10.002
Wooldridge, M., & Jennings, N. R. (1995). Agent theories, architectures, and languages: A survey*Springer Berlin Heidelberg.* Berlin, Heidelberg. Abstract retrieved from 10.1007/3-540-58855-8_1
Wray, S. (2010). How Pair Programming Really Works. *IEEE Softw., 27*(1), 50-55. doi:10.1109/ms.2009.199
Yilmaz, L., & Phillips, J. (2007). The impact of turbulence on the effectiveness and efficiency of software development teams in small organizations. *Software Process: Improvement and Practice, 12*(3), 247-265. doi:10.1002/spip.318
Zhang, H., Kitchenham, B., & Jeffery, R. (2009, 14-17 April 2009). Qualitative vs. Quantitative Software Process Simulation Modeling: Conversion and Comparison Symposium conducted at the meeting of the 2009 Australian Software Engineering Conference doi:10.1109/ASWEC.2009.45
Zhang, H., Kitchenham, B., & Pfahl, D. (2008, 3-5 Dec. 2008). Software Process Simulation Modeling: Facts, Trends and Directions Symposium conducted at the meeting of the 2008 15th Asia-Pacific Software Engineering Conference doi:10.1109/APSEC.2008.50
Zhang, H., Kitchenham, B., & Pfahl, D. (2010). *Software process simulation modeling: an extended systematic review*. Paper presented at the meeting of the Proceedings of the 2010 international conference on New modeling concepts for today's software processes: software process, Paderborn, Germany.
Zhen, Y., Wanpeng, Z., & Hongfu, L. (2018). *Artificial Intelligence Techniques on Real-time Strategy Games*. Paper presented at the meeting of the Proceedings of the 2018 2nd International

Conference on Computer Science and Artificial Intelligence, Shenzhen, China. doi:10.1145/3297156.3297188

Zhou, C., Kuttal, S. K., & Ahmed, I. (2018, 1-4 Oct. 2018). What Makes a Good Developer? An Empirical Study of Developers' Technical and Social Competencies Symposium conducted at the meeting of the 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) doi:10.1109/VLHCC.2018.8506577

Zieris, F., & Prechelt, L. (2014). *On knowledge transfer skill in pair programming*. Paper presented at the meeting of the Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy. doi:10.1145/2652524.2652529

Zualkernan, I. A., Darmaki, H. A., & Shouman, M. (2008, 16-18 Dec. 2008). A methodology for building simulation-based e-learning environments for Scrum Symposium conducted at the meeting of the 2008 International Conference on Innovations in Information Technology doi:10.1109/INNOVATIONS.2008.4781776